Technical Report TR-67-48 May 1967

# TRANSLATOR AND SIMULATOR for the COMPUTER DESIGN AND SIMULATION PROGRAM (CDSP) VERSION I.

by

Charles K. Mesztenyi
Senior Research Programmer
Computer Science Center

The work was performed at the Computer Science Center of the University of Maryland and was supported by Grant NsG-398 of the National Aeronautics and Space Administration.

Abstract

The design of a digital computer, or digital device, can be described by the Computer Design Language. This Language was proposed by Dr. Y. Chu: An ALGOL-like Computer Design Language, Comm. of ACM, Vol. 8, No. 10 (Oct. 1965), p. 607-615. This report describes a restricted form of the Language in more details for the use of the Computer Design and Simulation Program (CDSP) on IBM 7094. The CDSP consists of an Executive Main program under which various subprograms may be called. This report contains the detailed flow charts of the Translator and Simulator subprograms. The Translator establishes various symbol tables and a Polish string from the description of the designed computer. The Simulator executes test programs written for the designed computer and prints out the contents of prescribed registers.

Contents:

1. Computer Design Language

A Computer Design Language was proposed by Dr. Yaohan Chu: "An ALGOL-like Computer Design Language" Comm. of ACM, Vol. 8, No. 10 (Oct. 1965) p. 607-615. The Computer Design and Simulation Program (CDSP) implements a subset of this language on the IBM 7094 for simulation purposes. Using this language, the Design of a Digital Computer is described by the declaration of its digital devices and by the description of the interaction between the declared devices.

The following devices are permissible in CDSP, Version I.:

A Register is handled as a storage device identified by a name and by the number of bits it contains.

A Subregister refers to a portion of a register.

A Memory is a multiple storage device with its elements, called words, all having the same number of bits. The words are consecutively addressed, and only one of the words can be referenced at a time. The address of the referenced memory word is defined by the contents of a declared register.

A Decoder is an output signal (TRUE or FALSE). The subscript of the decoder is compared to the contents of a specified register or subregister defined in the decoder declaration and the output signal is given according to the equivalence of the two values. The TRUE or FALSE signal is represented with a binary bit one or zero, sometimes referred as YES or NO.

A Switch is handled as a special storage device, where the "contents" of the switch is one of its possible positions. Changing the position of a switch may initiate an action between other devices.

A Terminal represents output signals referring to the outcome of a logical expression simulating logical network with no storage element.

Clock represents a timing device giving an output signal at equal timesteps. During one timestep, different actions may occur between devices. During the simulation, the clock is also used for counting.

The interaction of the declared devices is described by Labeled Statements. These statements contain two parts: The Label part is a logical expression yielding one output signal (TRUE or FALSE). The other part, containing the microstatements, describes the action to be taken when the Label has a TRUE output signal.

Example:

```
REGISTER,  A(2), R, F(5), G(5)
DECODER,  K = A
CLOCK,  P
 ...
/K(3)*P/     F = G, R = 1, A = 0
 ...
/K(0)*P/     F = G*25, R = G(1), A = 7
 ...
```

In the first line, 4 registers, A, R, F and G are declared, they consist of 3, 1, 6 and 6 bits respectively.

The second line defines the decoder K connected to the register A.

Clock P is defined by the third line.

The last two lines are independent labeled statements. If the value of the label, K(3)*P, is TRUE, then the contents of G replaces the contents of F, R and A will have the contents 1 and 0 respectively. K(3)*P will be TRUE only if the contents of A is 3 and an output signal is given by the clock P. The second labeled statement is executed when K(0)*P is TRUE,

(i.e. the contents of A is zero) and an output signal is given by the clock P. In this case, the contents of F is replaced by the result of a logical AND with the contents of G and the octal constant 25, the contents of R is replaced by the second bit of G, and the contents of A will be 7.

The precise rules for forming the Declaration and Labeled Statements are described in the subsequent sections.

## 1.1 Card Format

Using the Computer Design Language, the machine design should be punched for CDSP with the following format:

Col. 1 normally will be blank.

If it contains "C", then it is regarded as a comment card, and it is not processed. If it contains "1", then it is regarded as a continuation of the last non-comment card.

Col. 2-72 contain the description statement. Blank characters are simply ignored by the Translator and may be used freely to improve the readability of the statements.

Col. 73-80 are not processed by the Translator, they may be used for sequence numbering or other identification.

## 1.2 Constants, Variables.

### 1.21 Constants

All numbers are regarded as positive octal integers, thus digits 8 and 9 may not be used for constants. Furthermore, no constants may exceed 12 octal digits. Examples: 17, 0, 123500

### 1.22 Variables

The Registers, Subregisters, Memories, Decoders, Switches, Clocks and Terminals are treated as variables. All variables must be declared in a Declaration Statement prior to their usage elsewhere.

The name of a variable must be unique and must consist of 1 to 6 alphabetic or numeric characters of which the first must be alphabetic.

Examples: A, BI9, R25ABC

Restriction: IF, THEN, END may not be used as variable names.

The usage of the variables after declaration may take different forms; they are described under the corresponding Declaration Statements.

## 1.3 Declaration Statements

All variables must be declared in one of the Declaration Statements prior to any use of that variable. Column 1 must be blank and col. 2-72 must contain the declaration type, comma and list. Blanks will be disregarded by the Translator.

### 1.31 Register Declaration

Form: REGISTER, list

Examples: REGISTER, A(5), B2(10),C

This statement declares all variables appearing in the list as registers. The list consists of single register definitions separated by commas. A single register definition consists of the name of the register followed by octal number enclosed in parentheses. This octal number, d, defines the register as one having d+1 bits which are numbered from zero to d consecutively. When a register consists of only one bit, the corresponding zero in parentheses may be deleted, see the above example with register C.

After its declaration, a register may be referred in the following forms:

1. B2
2. B2(3)
3. B2(0-2) or B2(4-7)
4. B2(ØP)

B2 refers to the contents of the full declared register.

B2(3) refers only to the 4th bit of the register.

B2(0-2) refers to the first 3 bits of the register

B2(ØP) refers to the part of the register defined by ØP as a subregister, see below:

1.32 Subregister Declaration

Form: SUBREGISTER, list

Example: SUBREGISTER, B2(ØP) = B2(1-4), A(ØP) = A(0-3), A(I) = A(4)

The subregister is always used with a register name, and it refers to a part of that register. Logically the list consists of a set of equivalences separated by commas. When B2(ØP) appears in the text following the above declaration example, B2(1-4) is understood.

1.33 Memory Declaration

Form: MEMORY, list

Example: MEMORY, M(C) = M(77,10), ND(J) = ND(6,3)

This declaration statement permits the user to establish various blocks of memory, each addressable by a specified address register. Each individual memory definition in the list is of the form

$$M(R) = M(d_1, d_2)$$

where

M is the name of the memory.

R is the name of the corresponding address register. R must have been previously declared.

$d_1$ is the dimension of the memory in octal. (i.e. the memory M consists of $d_1 + 1$ words consecutively numbered from 0 to $d_1$)

$d_2$ is the length of each word of the memory in octal. (i.e. each word of M consists of $d_2 + 1$ bits)

In later reference, a specific memory word may be identified by a constant address, e.g. M(12), or by its address register, M(R), in which case the contents of R defines the word address. In any case, one is always referring to the contents of the full word of memory.

1.34 Decoder Declaration

Form: DECODER, list

Example: DECODER, K = F, L = G(2-5)

The list consists of the individual decoder definitions separated by commas. An individual decoder definition consists of the name of the decoder, the = character and the name of a previously declared register which may be subscripted. Each decoder declaration provides a reference to the contents of the associated register or register-segment. In its use, the decoder name is always followed by a constant enclosed in parentheses, e.g. K(5), whose value is a single bit (1 or 0, TRUE or FALSE depending on whether or not the contents of the register or register part is equal to the given constant.

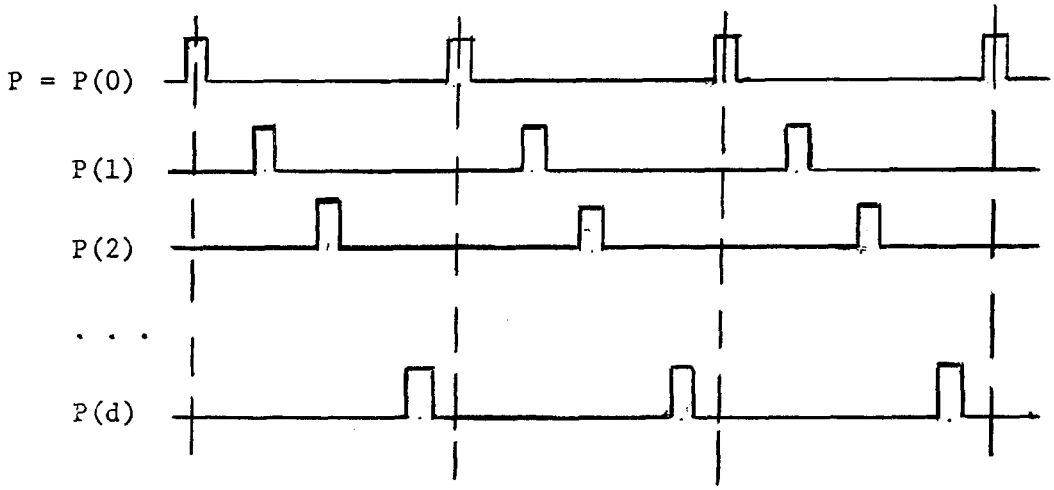
1.35 Clock Declaration

Form: CLOCK, P(d)

Example: CLOCK, P(2)

where P is the name of the clock and d is an octal number.

This declaration defines (d+1) (3 in the example) clocks. These clocks are referenced in the statements as P(0) (or simply P), P(1),..., P(d). The impulse diagrams for the clocks are assumed to be the following:



i.e. the time interval between the impulses given by the clocks are the same and the impulse given by clock P(i) immediately follows the impulse of P(i-1), i=1,2,...,d.

The clocks are used for counting during the simulation, thus the contents of the clock is the accumulated number of impulses given from the start.

1.36 Switch Declaration

Form: SWITCH, list

Example: SWITCH, START(OFF,ON), SENSE (P1, P2, P3)

The list consists of single switch definitions separated by commas. A single switch definition consists of the name of the switch followed by its positions enclosed in parentheses and separated by commas. The first position listed is assumed to be the starting position of the switch, i.e. the position of the switch at the start of the simulation. In the above example, the START switch is in OFF position, the SENSE switch is in $P_1$ position.

In later references, a switch is either checked for one of its position, or set to one of its position. When a switch is checked for a position, it has the form

NAME (POS) e.g. SENSE (P2)

and it gives a TRUE or FALSE (bit 1 or 0) depending on the position of the switch at the time of the reference.

When a switch is set to a position, it must appear as follows:

NAME = POS e.g. SENSE = P3

1.37 Terminal Declaration

Terminal declarations is described in 1.6.

## 1.4 Operators

With the exception of the complement operator, the available operators are binary operators, i.e. from two binary quantities they produce one binary quantity. In the Computer Design Language, a binary quantity is defined by its value and by the number of bits in which it is represented. The available binary operators require that the two binary quantities on which they operate, should have the same number of bits. E.g. the binary operator logical AND, operating on the contents of two registers, is used correctly if and only if the two registers have the same length in bits.

The octal constants used in the language are exceptions of the above rule by having undefined length. As soon as a constant is connected to a variable by a binary operator, its length is defined to be equal to the length of variable by taking the necessary number of bits from the binary representation of the constant from right to left, placing zero bits in the front of it if it is necessary. This definition requires that a binary operator may not be used to operate on two constants, i.e. at least one of the quantity must be a variable. Furthermore, the unary operator, taking the complement of the number, may not be used with a constant.

The following operators are available in CDSP, Version 1:

' (Apostrophe)

Example: A'

The apostrophe operator takes the binary complement of the number it refers to, the contents of A.

= (Equal sign)

Examples: A = B or A = 25

This is a replace operator i.e. the contents of A is replaced by the contents of B or by the constant 25.

+ (Plus sign)

Examples: A + B or A + 15

The plus sign denotes a logical OR between two binary quantities.

* (Asterisk)

Examples: A*B or A*27

The asterisk denotes a logical AND between two binary quantities.

Functions

Form: .NAME.

where NAME can consist of 1 to 6 alphabetic or numeric characters of which the first must be alphabetic, not including the preceding and following periods.

Examples: A.ADD.B or A.SHR.1

These examples are binary operators; they are supplied by the standard function package, see Section 5.

## 1.5 Expressions, Microstatements, IF Statement

### 1.51 Expressions

An expression is a proper sequence of constants, variables, operators (except =) and parentheses, with the usual mathematical meaning.

Examples: (A + B) * C + 1 or D or 10

An expression provides a single binary quantity (composed of one or more binary bits depending on the operators and variables involved) that result from the implied evaluation.

When the hierarchy of operations in an expression is not explicitly specified by the use of parentheses, it is understood by the Translator to be in the following precedence (from the innermost operations to the outermost):

| | |
|---|---|
| ' | Complementing |
| .xxx. | Functions |
| * | Logical AND |
| + | Logical OR |

1.52 Microstatement

A microstatement consists of a variable, the replace operator "=" and an expression.

Examples: A = 1 or A = B * C + D

1.53 IF Statement

The IF Statement has the following form:

IF (expression) THEN (microstatements and/or IF statements)

The expression enclosed in parentheses must give a single bit answer one or zero (TRUE or FALSE). If the answer is TRUE, all microstatements separated by commas and enclosed in parentheses after THEN will be evaluated. If the answer is FALSE, then all statements within the parentheses are skipped.

Example: IF ((G(1)*B(1)).EQU.1) THEN (G(1) = 0, A = B)

The IF Statement is sometimes referred as conditional microstatement.

1.6 Terminal Declaration

Form: TERMINAL, list

Example: TERMINAL, T = A * B + C, R = G(2-7) + 1

The list consists of the individual terminal definitions separated by commas. The individual terminal is defined by a microstatement, i.e. by the name of the terminal followed by the replace operator and by an expression. All variables appearing in the expression must have been declared previously.

When a terminal is referenced, its value is the binary quantity obtained by the evaluation of its declared expression.

1.7 Comment Statement

When column 1 contains a "C", the card is understood and it will not be processed.

1.8 Switch Statement and Labeled Statement

Switch Statements and Labeled Statements are logically different, but their form is the same. Either consists of a label which is an expression enclosed in slashes, and one or more microstatements following the label. The microstatements are separated by commas. The end of the statement is indicated either by a new label or by the END statement, thus it may be written in more than one line (card) without using col. 1 =1 for indicating continuation.

Form:

/Expression/ microstatements and/or IF statements

In both types, the expression of the label must give a single bit value, TRUE or FALSE.

1.81 Switch Statement

The label of the Switch Statement is a switch name followed by one of its positions enclosed in parentheses:

/Name (Position)/

During the simulation, the switch statement has the following meaning:

If the named switch is set to the position indicated within the parentheses, the label expression is TRUE and the following microstatements will be executed.

The position of a switch may be set by either a microstatement (NAME = POS), which corresponds to an internal switch operation, or by a manual operation which is simulated by reading a control card.

1.82 Labeled Statement

During the simulation, all labels are evaluated. If there are more than one label with a TRUE value, the simulation stops indicating an error. If there is only one label with TRUE value, then the microstatements following that label will be executed and the process repeats itself. If there are no labels with TRUE value, the simulation gives the control back to the executive routine.

1.9 END statement

The physical end of the program describing the designed computer is indicated by the END statement. The word END can be anywhere in col. 2-72, with the rest of the columns left blank.

## 2. EXECUTIVE PROGRAM

The Computer Design Program contains various subprograms, such as the Translator, Simulator, Boolean and Circuit Subprograms. The subprograms are monitored by the Executive Program. It accepts control cards with $ character in column 1 and it gives the control to the proper subprogram.

Beside monitoring, the Executive Program also handles the communication between different subprograms. The results of the executed subprograms are saved on an auxiliary tape. The contents of this tape is referenced through a list in the Executive Program which is called the Communication Table. When another subprogram needs the results of a previously executed subprogram, the Executive Program supplies them by reading the auxiliary tape.

The Executive Program has the following control cards:

| | |
|---|---|
| $START | causes an overall start with empty Communication Table. |
| $START m | causes a restart. It must be followed by m cards containing the Communication Table. |

# 3. TRANSLATOR

## 3.1 Description

The Translator establishes various symbol tables and a Polish string from the input cards containing the description of the designed computer using the Computer Design Language.

The established tables are as follows:

| NAME | COUNTER | DESCRIPTION | |
|---|---|---|---|
| R | NR | Register Table | Symbol Tables |
| SR | NSR | Subregister Table | |
| M | NM | Memory Table | |
| D | ND | Decoder Table | |
| SW | NSW | Switch Table | |
| T | NT | Terminal Table | |
| C | NC | Clock Table | |
| SWL | NSWL | Switch Lable Table | |
| LST | NLST | Labeled Statement Table | |
| P | NP | Polish String | |
| STORE | NSTORE | Storage Array.<br>The counter corresponds to the last bit used in the array. | |

The Symbol Tables are generated by the corresponding Declaration Statement. For each storage element of the designed computer, such as the registers, memories and switch positions, a unique position is assigned in the Storage Array.

The logical expressions and microstatements, such as the Terminal Expressions, the microstatements of the Switch Label Statement, the Labeled Statements, are translated into reverse Polish notation and stored in the Polish String. This Polish String will be used by the simulator to evaluate the expressions and microstatements. The Terminal Table contains the pointers to their expression in the Polish String. The Switch Label Table contains the labels (switch names and positions) with the pointers to their microstatements in the Polish String. The Labeled Statement Table contains the pointers to the label expressions in the Polish String.

Register Table, R(i,j), i=1,2,3; j=1,..., NR

| R(1,j) | R(2,j) | R(3,j) |
|---|---|---|
| BCD name of the register | Number of bits of the register | Location of the last bit of the register in the Storage Array |

Subregister Table, SR(i,j), i=1,2,3,4; j=1,..., NSR

| SR(1,j) | SR(2,j) | SR(3,j) | SR(4,j) |
|---|---|---|---|
| BCD name of the register | Number of bits of the subregister | Location of the last bit of the Subregister in the Storage Array | BCD name of the corresponding register |

Memory Table, M(i,j), i=1,2,3,4,5; j=1,..., NM

| M(1,j) | M(2,j) | M(3,j) | M(4,j) | M(5,j) |
|---|---|---|---|---|
| BCD name of the memory | Number of bits per word of the memory | Location of the last bit of the first word of the memory in the Storage Array | Index k of the corresponding address register R(1,k) | Number of words in the memory |

Decoder Table, D(i,j), i=1,2,3; j=1,..., ND

| D(1,j) | D(2,j) | D(3,j) |
|---|---|---|
| BCD name the decoder | Number of bits in the corresponding register | Location of the last bit of the corresponding register in the Storage Array |

Switch Table, SW(i,j), i=1,2,3; j=1,..., NSW

| SW(1,j) | SW(2,j) | SW(3,j) |
|---|---|---|
| BCD name of the switch | Number of positions | Location of the last bit of the current position in the Storage Array |

Format of the Storage Array with Switches:

| | 36 bits Current position | 36 bits 1st position | 36 bits 2nd position | ... | 36 bits Sw(2,j)th position (BCD) | |
|---|---|---|---|---|---|---|

SW(3,j)

Terminal Table, T(i,j), i=1,2; j=1, ..., NT

| T(1,j) | T(2,j) |
|---|---|
| BCD name of the terminal | Entry point to the expression in the Polish String |

Clock Table, C(i), i=1,..., NC+2

| C(1) | C(2) | C(3) | C(4) | ... | C(NC+1) | C(NC+2) |
|---|---|---|---|---|---|---|
| BCD name of the clock | NC=number of clocks | Count for 0-clock | Count for 1st clock | ... | Count for (NC-2)th clock | Count for (NC-1)th clock |

Switch Label Table, SWL(i,j), i=1,2,3,j=1,..., NSWL

This table contains the necessary information about the Labels of the Switch Statements. It can be rapidly scanned if a switch interrupt occurs by a manual or internal switch setting.

| SWL(1,j) | SWL(2,j) | SWL(3,j) |
|---|---|---|
| BCD name of the Switch | BCD name of the position of the Switch | Entry index to the microstate-ments in the Polish String |

Switch Label Table, SWL(i,j), i=1,2,3,j=1,..., NSWL

This table contains the necessary information about the Labels of the Switch Statements. It can be rapidly scanned if a switch interrupt occurs by a manual or internal switch setting.

| SWL(1,j) | SWL(2,j) | SWL(3,j) |
|---|---|---|
| BCD name of the Switch | BCD name of the position of the Switch | Entry index to the microstatements in the Polish String |

Labeled Statement Table, LST(j), J=1,..., NLST

All Labeled Statements are consecutively numbered from one to NLST. This table contains the corresponding entry indices to their Label expression in the Polish String.

| LST(j) |
|---|
| k = index P(1,k) as entry point to the Label of j$^{th}$ Statement |

Polish String, P(1,j), P(2,j); j=1,..., NP

As it was noted before, the microstatements and label expressions are stored in reverse Polish notation in the Polish String. The rules of the translation of the expressions is described by C. L. Hamblin: Translation to and from Polish notation. Comput. J. 5, 3(Oct. 1962), 210-213 in the "I. Orthodox A to Reverse Polish" Section. A position in the Polish String consisting of two words (P(1,j) and P(2,j)) may refer to a storage element,

to a constant, to an operator or to a special instruction. The first word, P(1,j) defines the above types.

Storage elements: P(1,j) is a positive non-zero integer and it refers to the number of bits of the storage element.

Register, Subregister, Switch name, Switch position

| P(1,j) | P(2,j) |
|---|---|
| Number of bits of the storage element | Location of its last bit in the Storage Array |

The switch name corresponds to the current position location.

Memory Word

When a memory word with constant address is referenced in the text, its translated form will be the same in the Polish String, as a register. When a memory word with its address register is referenced, it is translated into two positions of the Polish String using the following form:

| P(1,j) | P(2,j) | P(1,j+1) | P(2,j+1) |
|---|---|---|---|
| Number of bits in the memory word | Location of the last bit of the address register in the Storage Array with negative sign | Number of bits in the address register | Location of the last bit of the first word of the memory in the Storage Array |

Note that the distinction is made between the two forms by the sign of P(2,j).

Constants

Constants are represented directly in the Polish String. This restricts the constant with maximum 36 bits, but the actual size in number of bits is not defined by the Translator. Since a constant is always connected with a storage element through a binary operator, the size of the constant is defined by that storage element and the binary operator. For this reason constants are represented with P(1,j)=0.

| P(1,j) | P(2,j) |
|---|---|
| 0 | The constant as a full word with preceding zeros. |

Operators

All unary and binary operators are represented with P(1,j) = -101. In later versions, we might separate the different operators with P(1,j) = -101, -102, -103,... if that gives any special advantage for other subprograms.

| P(1,j) | P(2,j) |
|---|---|
| -101 | BCD name of the operator |

Example:

A Decoder is translated as an expression. Thus if D is the decoder of register F which has n(F) bits and location L(F) in the Storage Array, the notation

D(15)

has the equivalent form in the Polish String

| k | P(1,k) | P(2,k) |
|---|---|---|
| j | n(F) | L(F) |
| j+1 | 0 | 15 |
| j+2 | -101 | EQU |

The replace operator (= character) is regarded as a special operator. The form of the Polish String for

A=(Expression)

is the following:

| P(1,k) | P(2,k) |
|---|---|
| (Expression translated) | |
| -101 | = |
| n(A) | L(A) |

Special Instructions

Special Instructions consist of the following:

1. Clock reference
2. Entry
3. Exit
4. Transfers

1. Clock reference

When a clock name P appears in an expression, it is translated into the following form

| P(1,j) | P(2,j) |
|---|---|
| -1 | k |

where k is the index value the clock was referenced, P(k). If it was not indexed, then k = 0.

2-3. Entries and Exists

Since during the Simulation, the Polish String is executed in segments, we define a full segment of the Polish String as follows:

A Full Segment of the Polish String are consecutive positions such that

1. The first position is an Entry position
2. The last position is an Exit position
3. Between the first and last positions there are no Entry and Exit positions, and only one of the following four cases is possible:
   a) translated form of the expression of one terminal
   b) translated form of one switch statement
   c) translated form of the expression of the label of one labeled statement
   d) translated form of all the microstatements of one labeled statement.

The format of the Entry and Exit positions are as follows:

Entry

| P(1,j) | P(2,j) |
|---|---|
| -2 | j* |

Exit

| P(1,j*) | P(2,j*) |
|---|---|
| -3 | 0 |

where P(2,j) contains the index value J* of the corresponding Exit position. The Full Segment of the Polish String enclosed by the Entry and Exit positions can be regarded as a subroutine and P(2,j*) will be used for return transfer.

4. Transfers

Subroutine Transfer

When a terminal is represented in an expression E, it is necessary to evaluate the full segment of the expression of the terminal inside of the full segment of the expression E. This is accomplished by the Subroutine Transfer which has the following form

| P(1,j) | P(2,j) |
|---|---|
| -5 | k |

Position k should be in the same full segment with j.

Example

Consider the following statements.

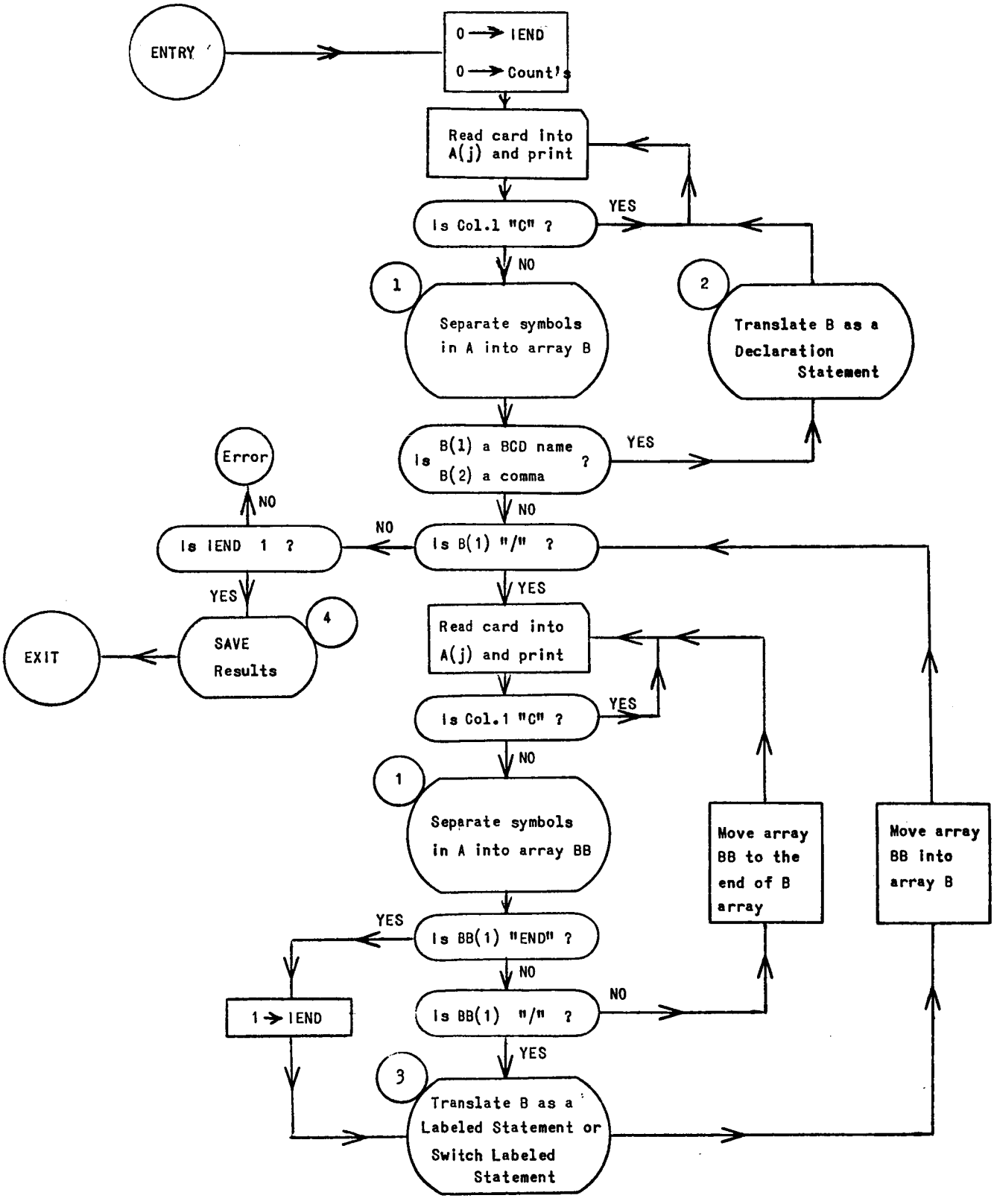
```
REGISTER,     A(5), B(5), C, D(5), E(5), G(3), F(3)
DECODER,      K(17) = F
CLOCK,        P(2)
TERMINAL,     SUB = K(10)*C*P(1)
/SUB*E(1)/    IF (G(2).EQU.1) THEN (A = B), D = A+E
     . . .
```

If L(x) indicates the location of register x in the Storage Array, then the Polish Array has the following segments (j indices were arbitrarily chosen for this illustration):

| j | P(1,j) | P(2,j) | |
|---|---|---|---|
| 70 | 2 | 78 | Terminal Segment |
| 71 | 4 | L(F) | |
| 72 | 0 | 10 | |
| 73 | -101 | EQU | |
| 74 | 1 | L(C) | |
| 75 | -101 | * | |
| 76 | -1 | 1 | |
| 77 | -101 | * | |
| 78 | -3 | 0 | |
| . | . | . | |
| . | . | . | |
| . | . | . | |
| 120 | -2 | 124 | Label Segment |
| 121 | -4 | 70 | |
| 122 | 1 | L(E(1)) | |
| 123 | -101 | * | |
| 124 | -3 | 0 | |
| 125 | -2 | 138 | Microstatements Segment |
| 126 | 1 | L(G(2)) | |
| 127 | 0 | 1 | |
| 128 | -101 | EQU | |
| 129 | -5 | 133 | |
| 130 | 6 | L(B) | |
| 131 | -101 | = | |
| 132 | 6 | L(A) | |
| 133 | 6 | L(A) | |
| 134 | 6 | L(E) | |

| j | P(1,j) | P(2,j) |
|---|---|---|
| 135 | -101 | + |
| 136 | -101 | = |
| 137 | 6 | L(D) |
| 138 | -3 | 0 |
| 139 | ... | ... |

## 3.2 TRANSLATOR - FLOW CHART



(1) Separate Symbols from array A into array B

SUBROUTINE SEPT (A, N, B, IND, M)

Given 6xN BCD characters packed in the array A(k), K = 1, ..., N, the routine unpacks the characters and stores them in the array B(j), j = 1, ..., M. The array IND (j), j = 1, ..., M will contain identifying integers as follows:

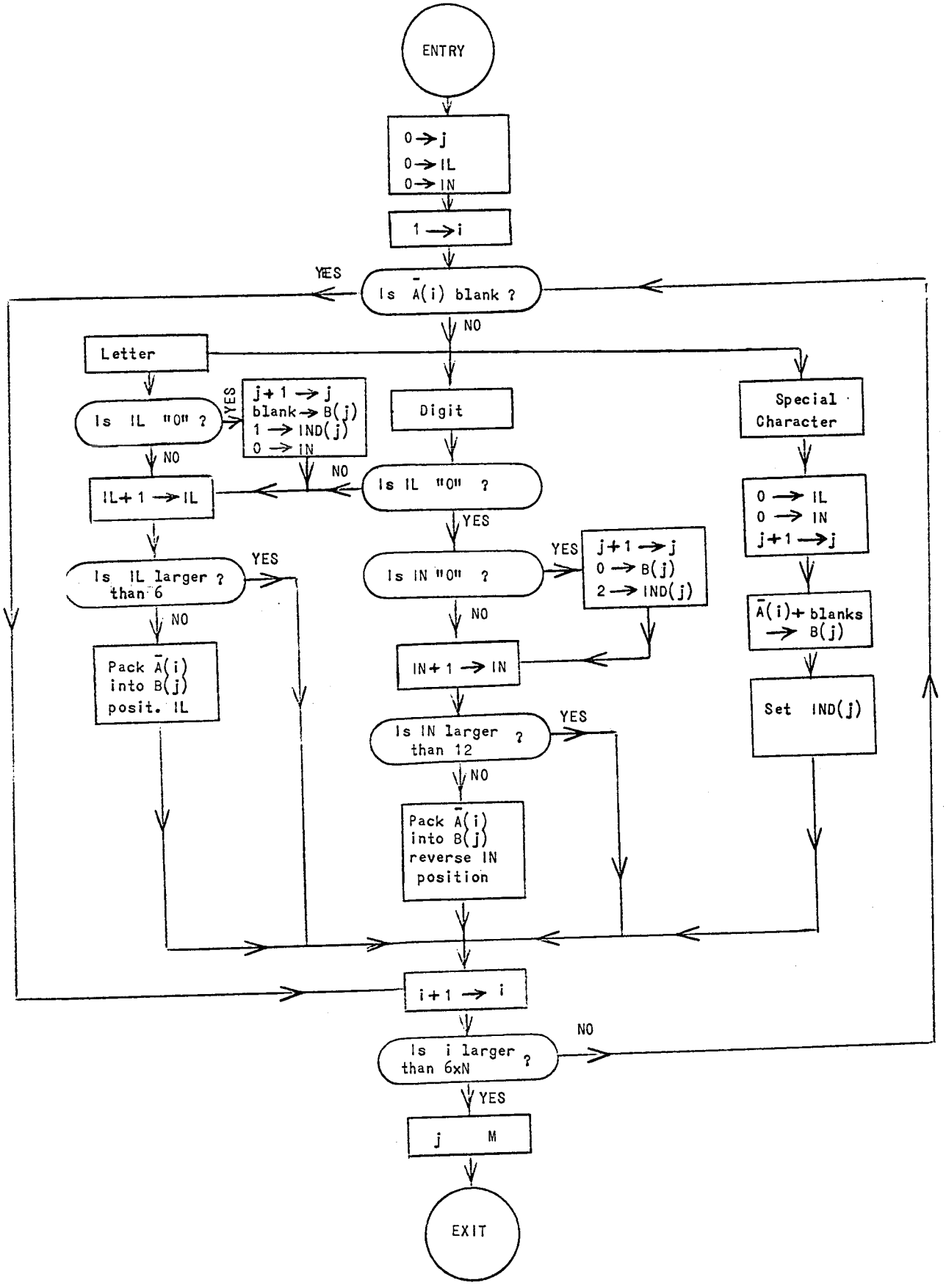
| IND (j) | B (j) | |
|---|---|---|
| 1 | Variable name | |
| 2 | Number | (constant) |
| 3 | ( | (left parenthesis) |
| 4 | ) | (right parenthesis) |
| 5 | . | (period) |
| 6 | = | (equals) |
| 7 | / | (slash) |
| 8 | * | (asterisk) |
| 9 | + | (plus) |
| 10 | - | (minus) |
| 11 | $ | (dollar sign) |
| 12 | , | (comma) |
| 13 | ' | (apostrophe) |

The unpacking is done as follows:

1. Blanks are disregarded.
2. Each character which is not a letter or digit is regarded as a separator and it occupies one word in the array B, left adjusted with 5 blanks followed.

3. A string of letters or digits are regarded as a Variable Name if the first character is a letter. If it consists of less than 6 characters, they will be left adjusted in B(j) with blanks followed. If it consists of more than 6 characters, only the first 6 characters will be retained in B(j).
4. A string of digits (from 0 to 7) is regarded as an Octal Number. If the string has less than 12 digits, the octal number is placed in B(j) right adjusted. If it has more than 12 digits, only the first 12 digits will be retained.
5. Digits 8 and 9 are not allowed.

## FLOW - CHART


ENTRY
0 → j
0 → IL
0 → IN
1 → i
YES
Is Ā(i) blank ?
NO
Letter
Digit
Special Character
Is IL "0" ?
YES
j+1 → j
blank → B(j)
1 → IND(j)
0 → IN
NO
IL+1 → IL
Is IL "0" ?
NO
YES
Is IL larger ? than 6
YES
NO
Pack Ā(i) into B(j) posit. IL
Is IN "0" ?
YES
j+1 → j
0 → B(j)
2 → IND(j)
NO
IN+1 → IN
Is IN larger ? than 12
YES
NO
Pack Ā(i) into B(j) reverse IN position
0 → IL
0 → IN
j+1 → j
Ā(i)+ blanks → B(j)
Set IND(j)
i+1 → i
Is i larger than 6xN ?
NO
YES
j M
EXIT

(2) Translate B as a Declaration Statement

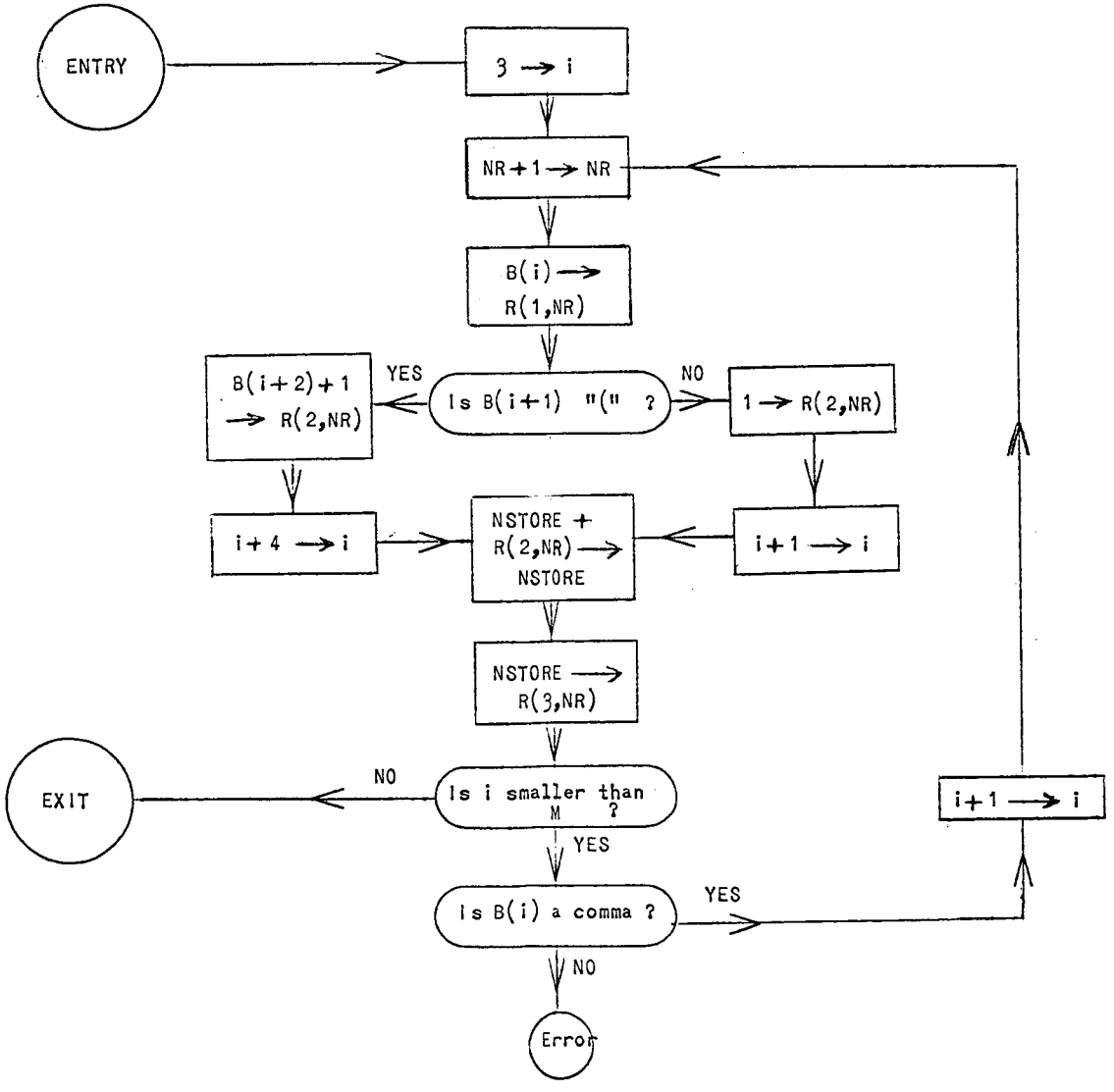
B(1) defines the type of declaration. M is the number of symbols separated in the array B. The following flow-charts are given by types:

REGISTER, list

B(1) = REGIST

The list consists of one or more registers in the form of

NAME ($d_1$) or NAME



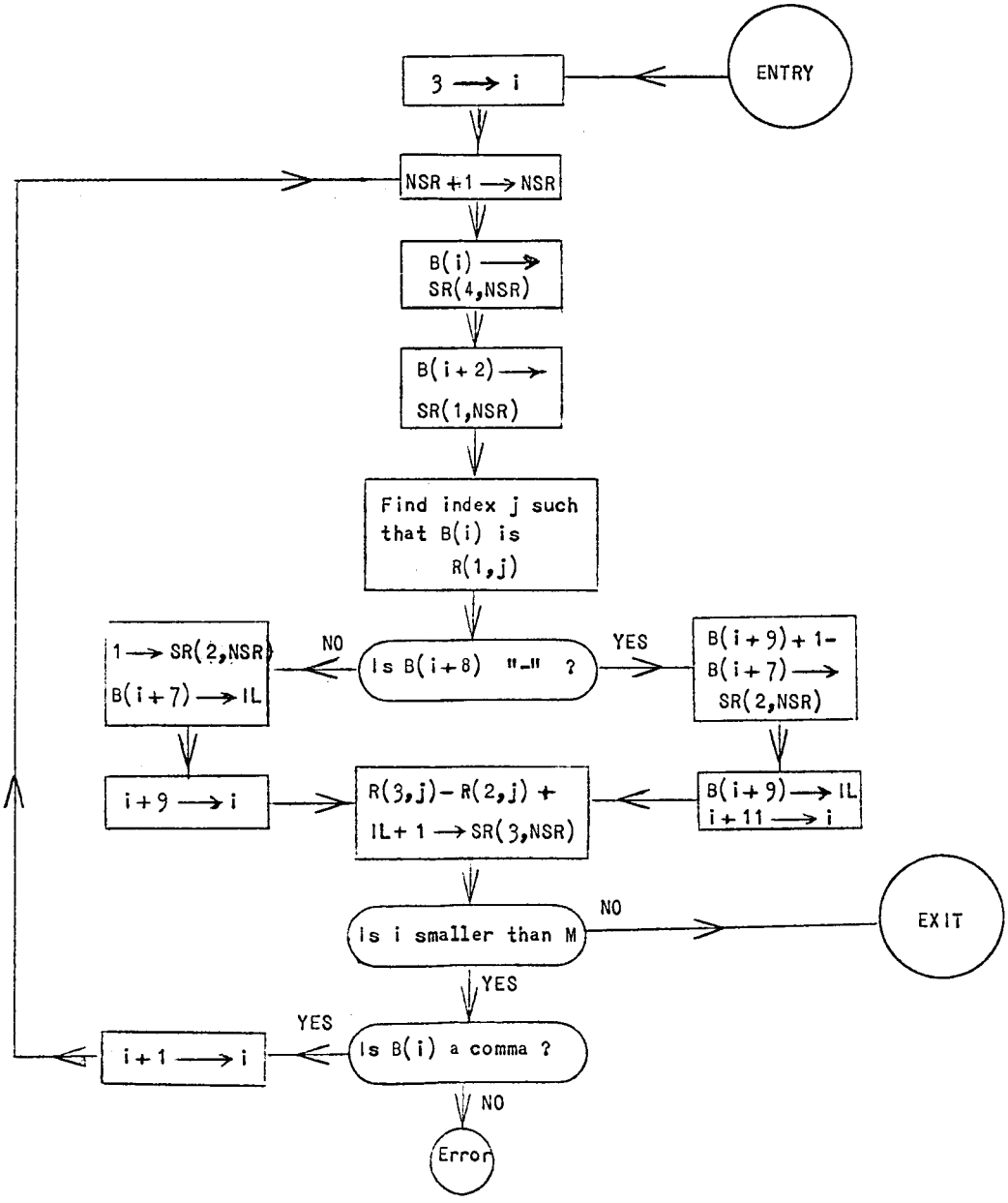
SUBREGISTER, list

B(1) = SUBREG

The list consists of one or more subregisters in the form of

$$R(S) = R(d) \quad \text{or} \quad R(S) = R(d_1 - d_2)$$

where R is the name of a register, S is the name of the subregister.

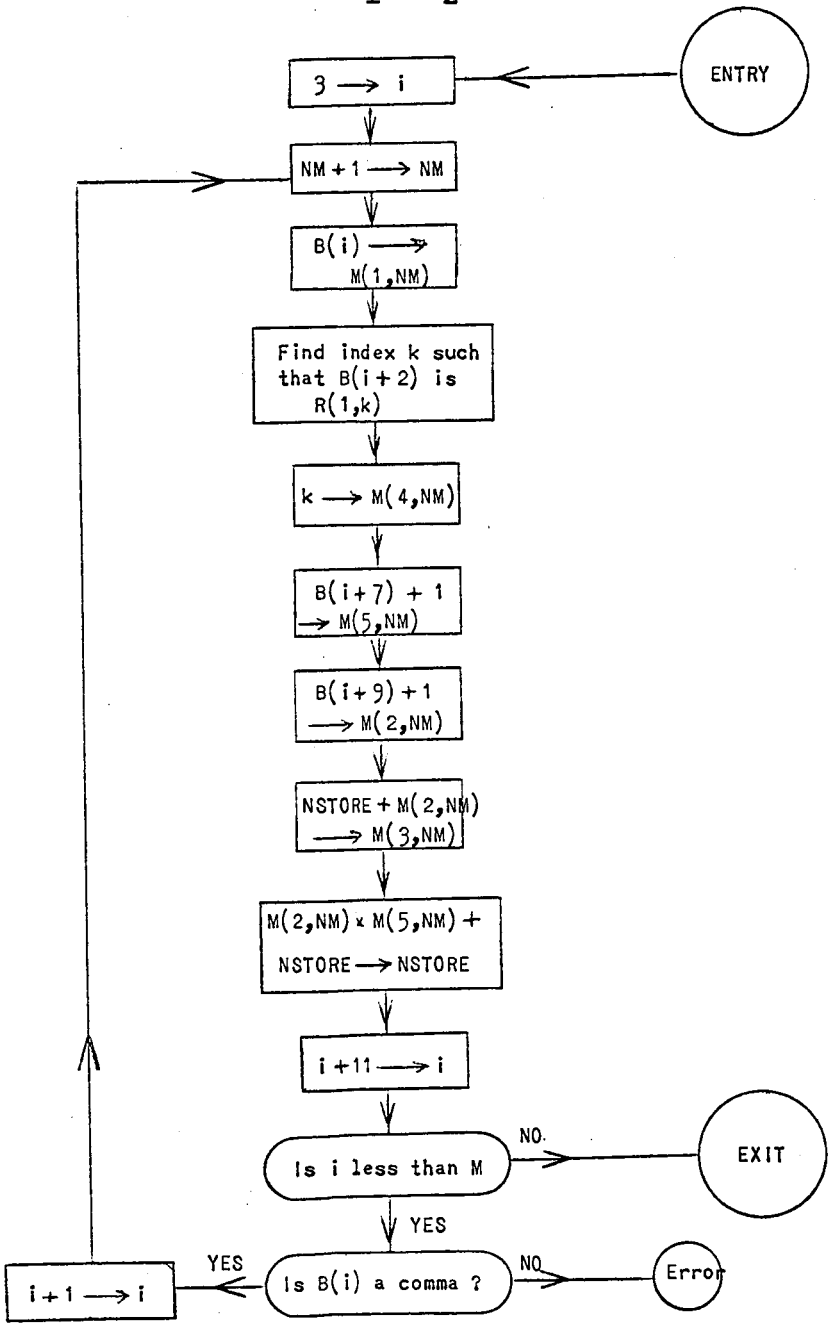


MEMORY, list

B(1) = MEMORY

The list consists of one or more memories in the form of

NAME (R) = NAME ($d_1$, $d_2$)



DECODER, list

B(1) = DECODE

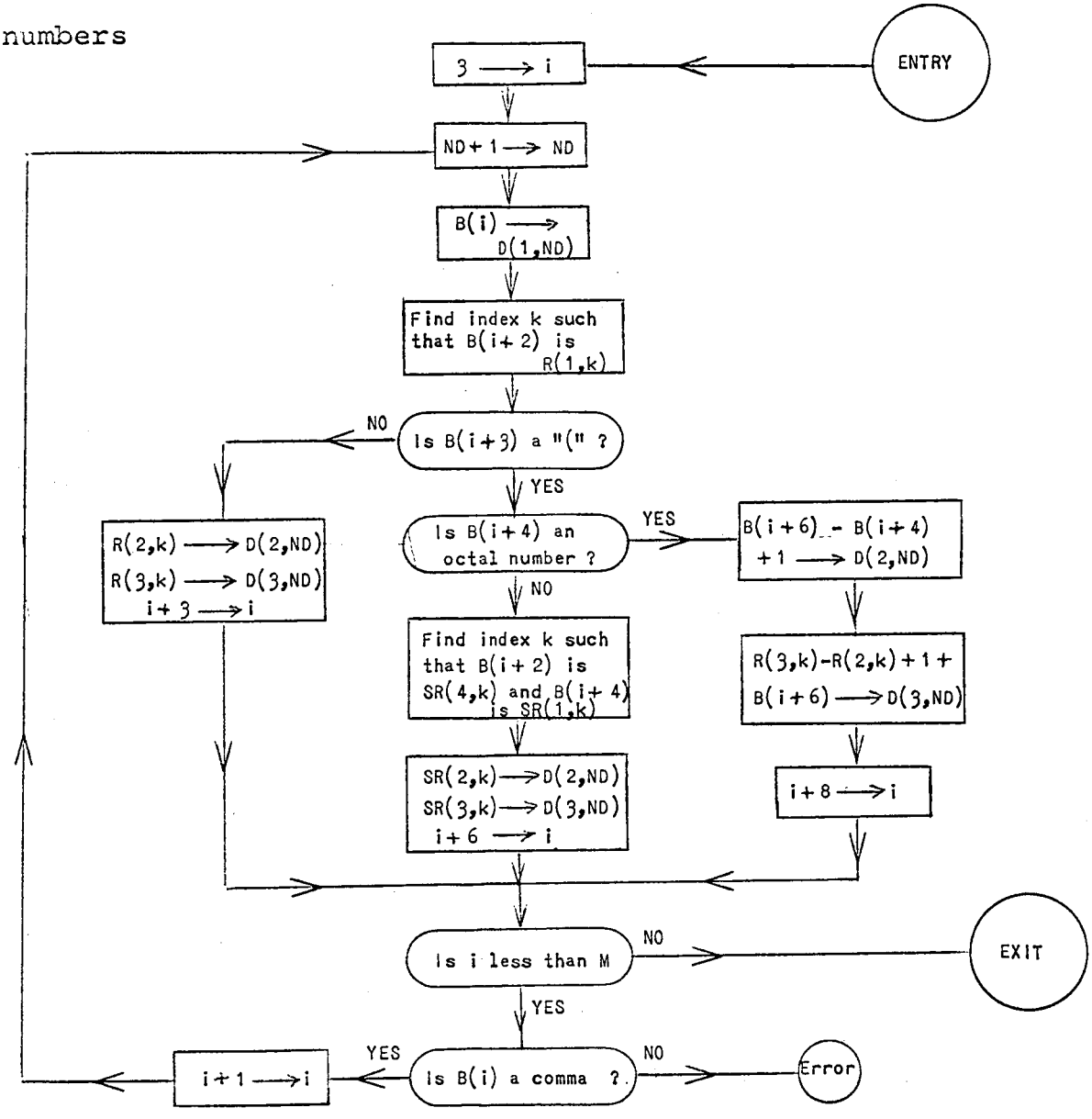
The list consists of one or more decoders in the form of

NAME = R

NAME = R(S)

NAME = R($d_1$ - $d_2$)

where R is a register, S is a subregister, $d_1$ and $d_2$ are octal numbers

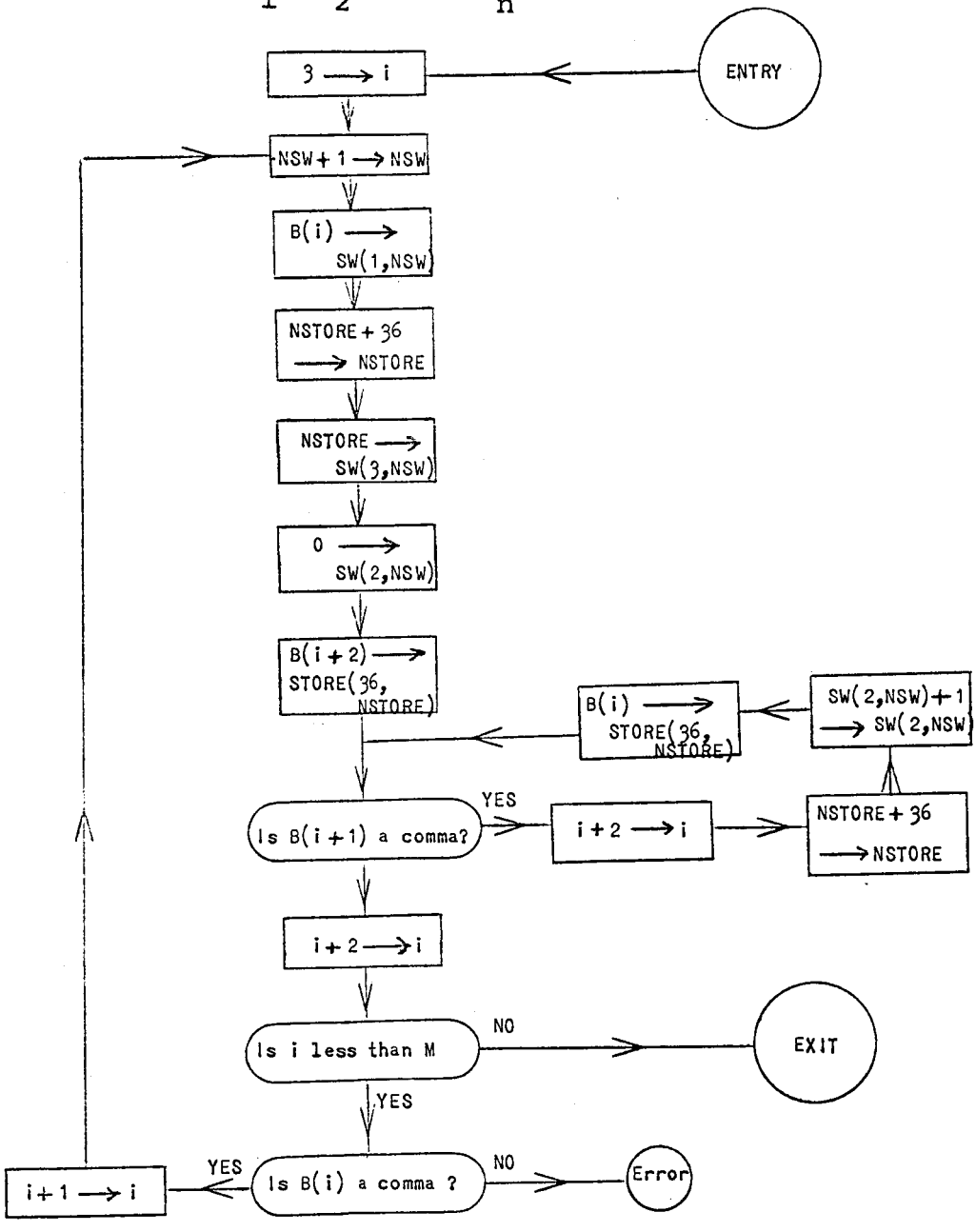


SWITCH, list

B(1) = SWITCH

The list consists of one or more switches in the form of

NAME ($P_1$, $P_2$, ..., $P_n$)



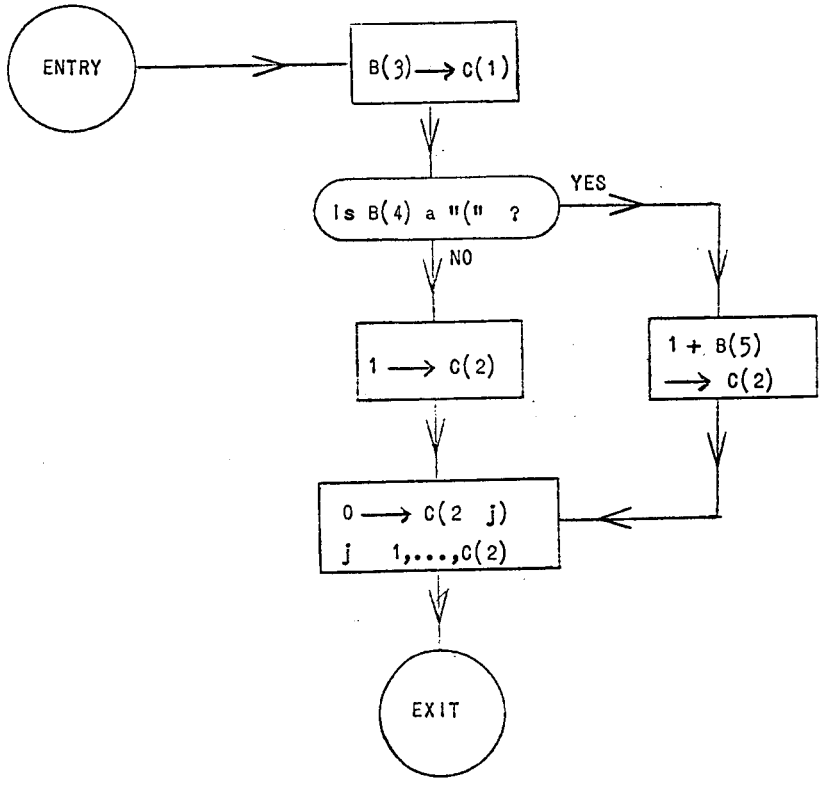
CLOCK, list

B(1) = CLOCK

The list has the form of NAME

or

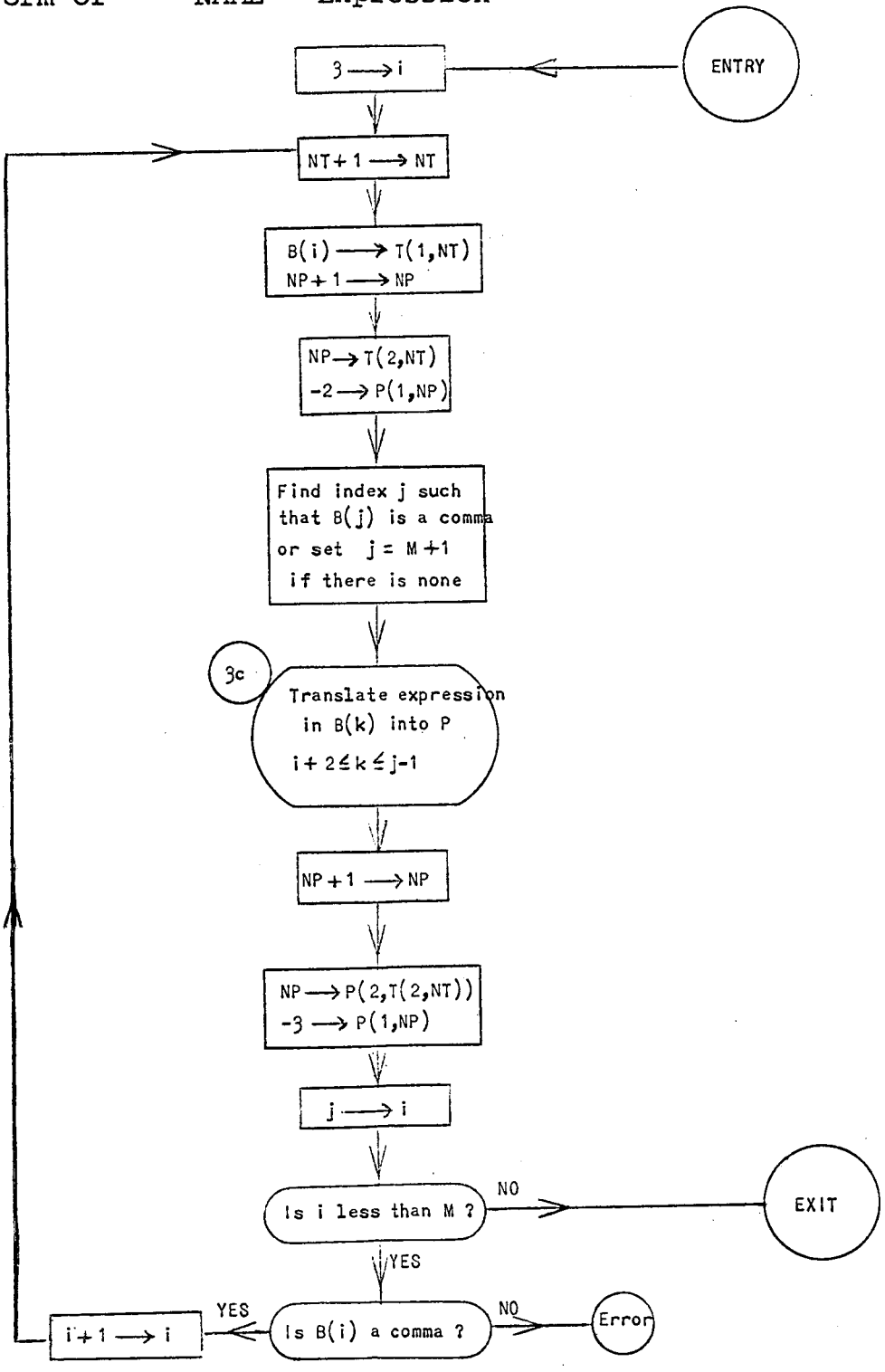
NAME (d)



TERMINAL, list

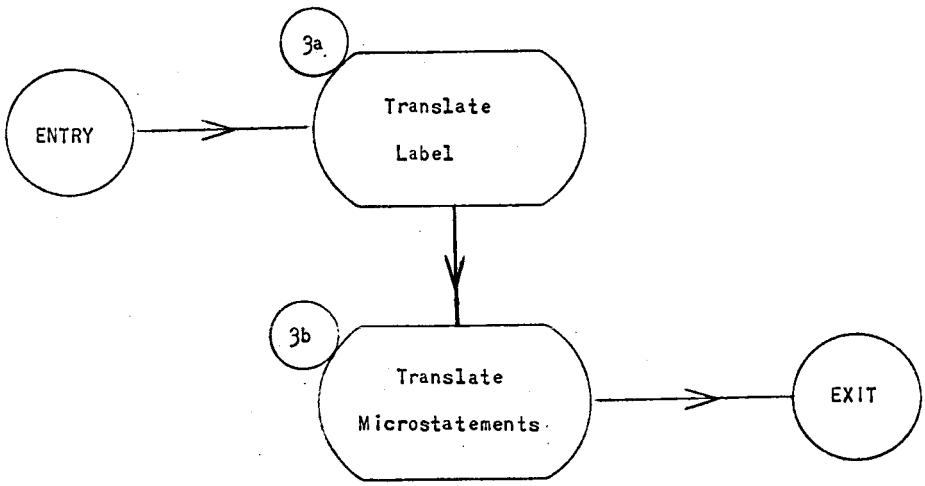
B(1) = TERMIN

The list consists of the individual terminal declarations in the form of NAME = Expression



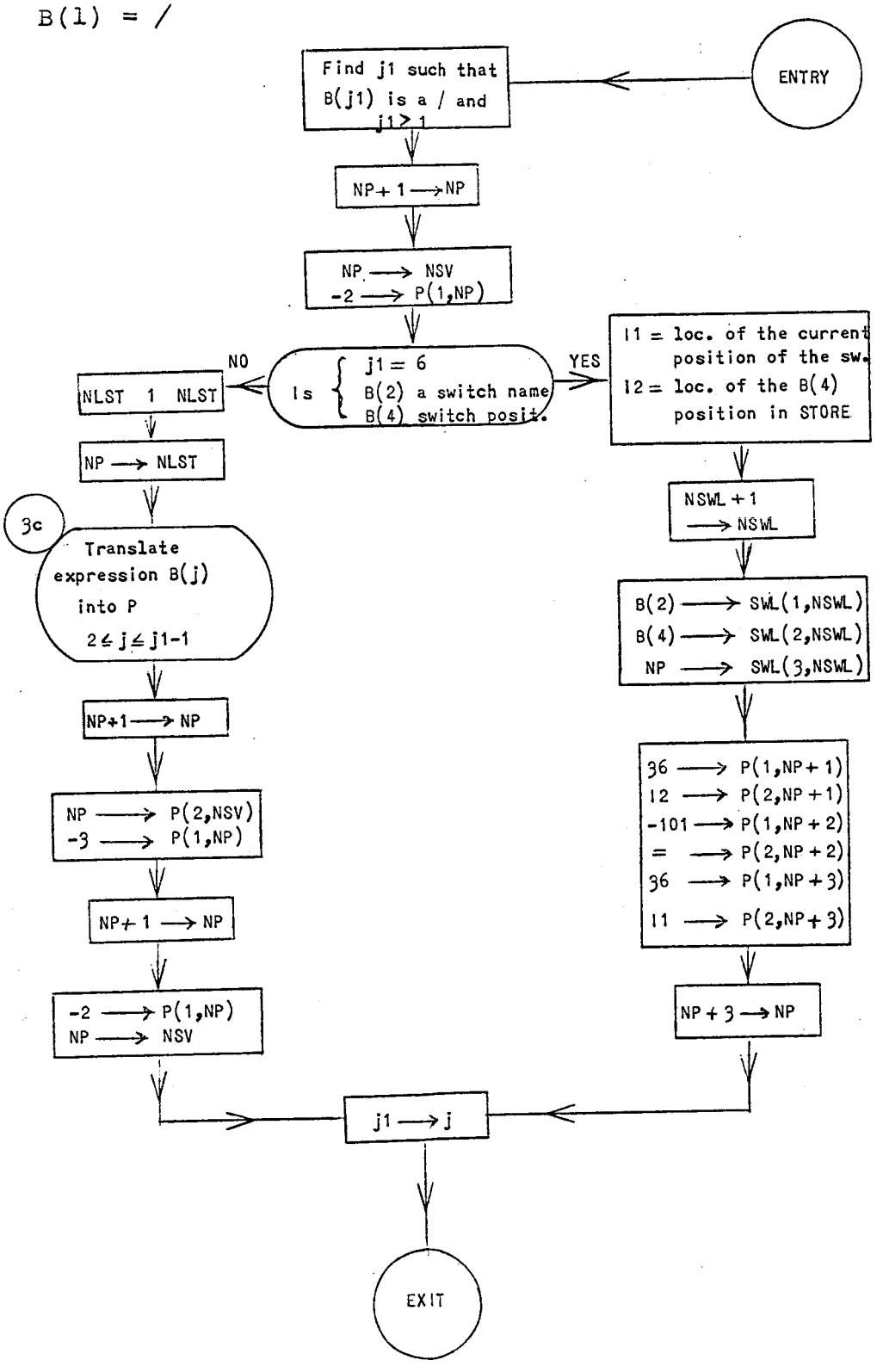
(3) Translate B as a Labeled Statement or a Switch Statement

The only difference between a Labeled Statement and the Switch Statement is in the translation of the Label. The translation of the microstatements are the same.



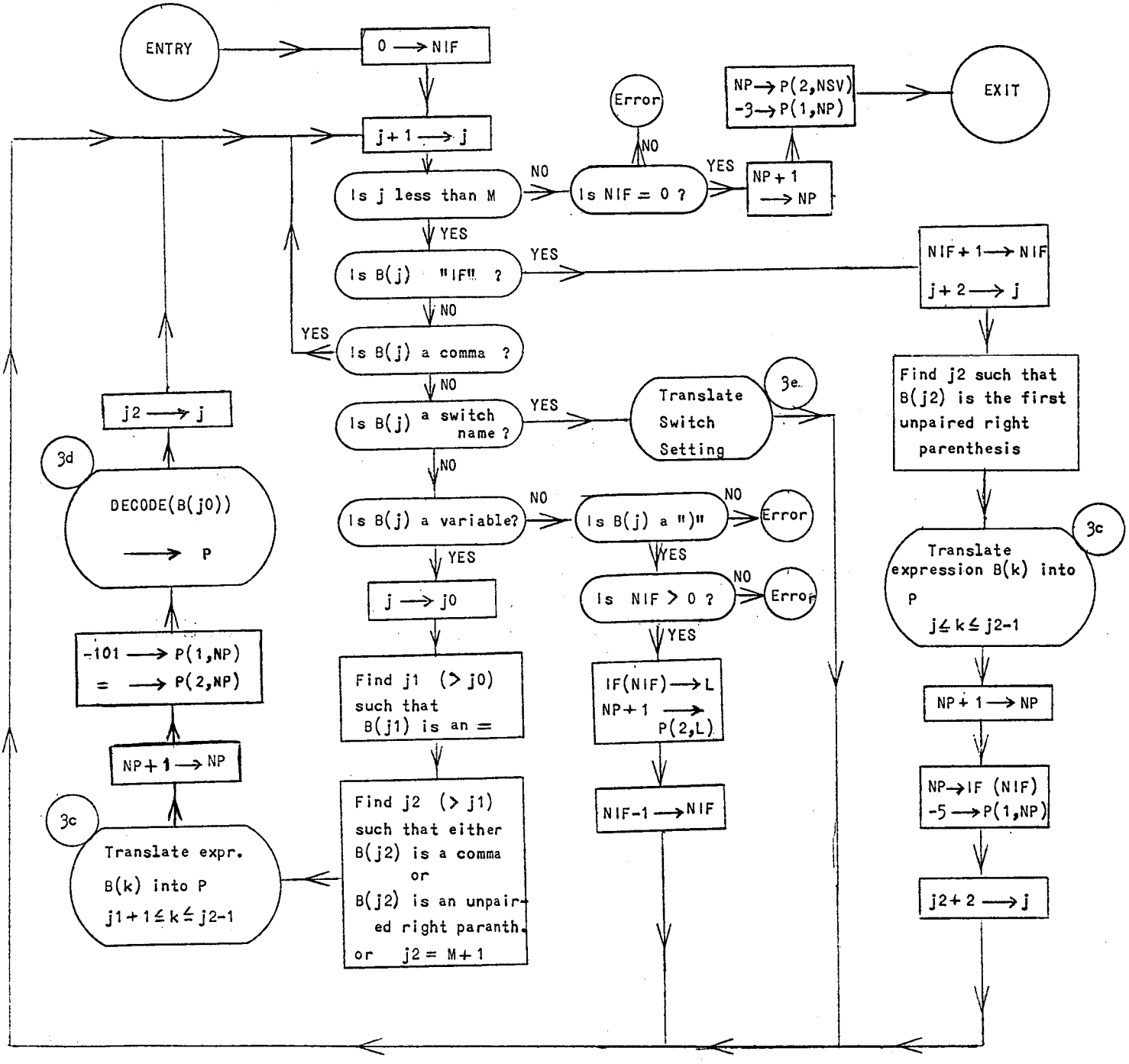
(3a) Translate Label

B(1) = /



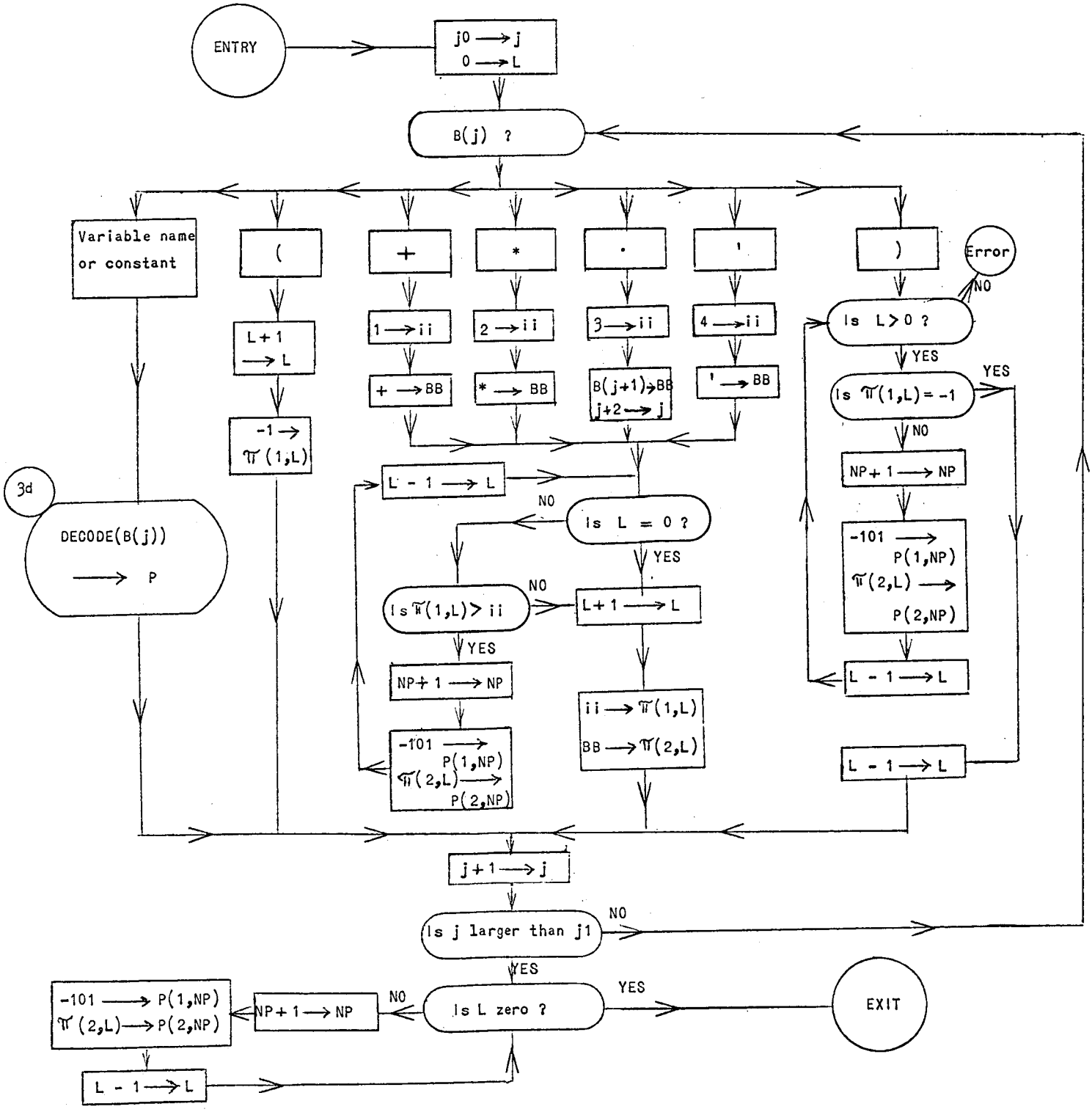
(3b) Translate Microstatements

NIF = depth of the IF conditions. The array IF (L), L = 1, ..., NIF contains the transfer points in the Polish String.



(3c) Translate expression

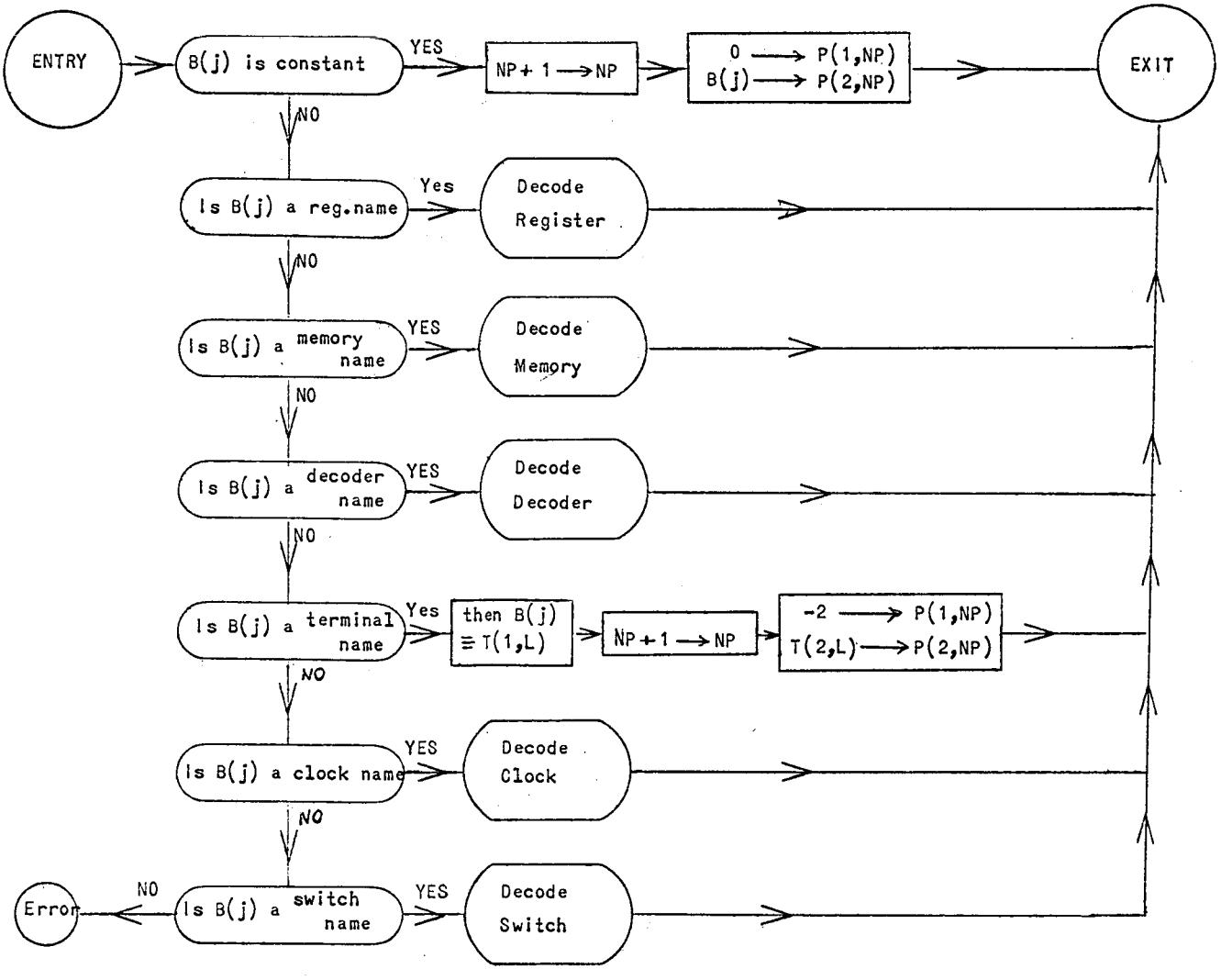
The expression is contained in B(j), j = $j_0$, ..., $j_1$ and it is translated into the Polish array P($\frac{1}{2}$, k), K = NP + 1, ... $\pi(\frac{1}{2}, L)$ is a temporary array.



(3d) DECODE (B(j)) → P

B(j) must be one of the following:

1. Constant
2. Register name
3. Memory name
4. Decoder name
5. Terminal name
6. Clock name
7. Switch name



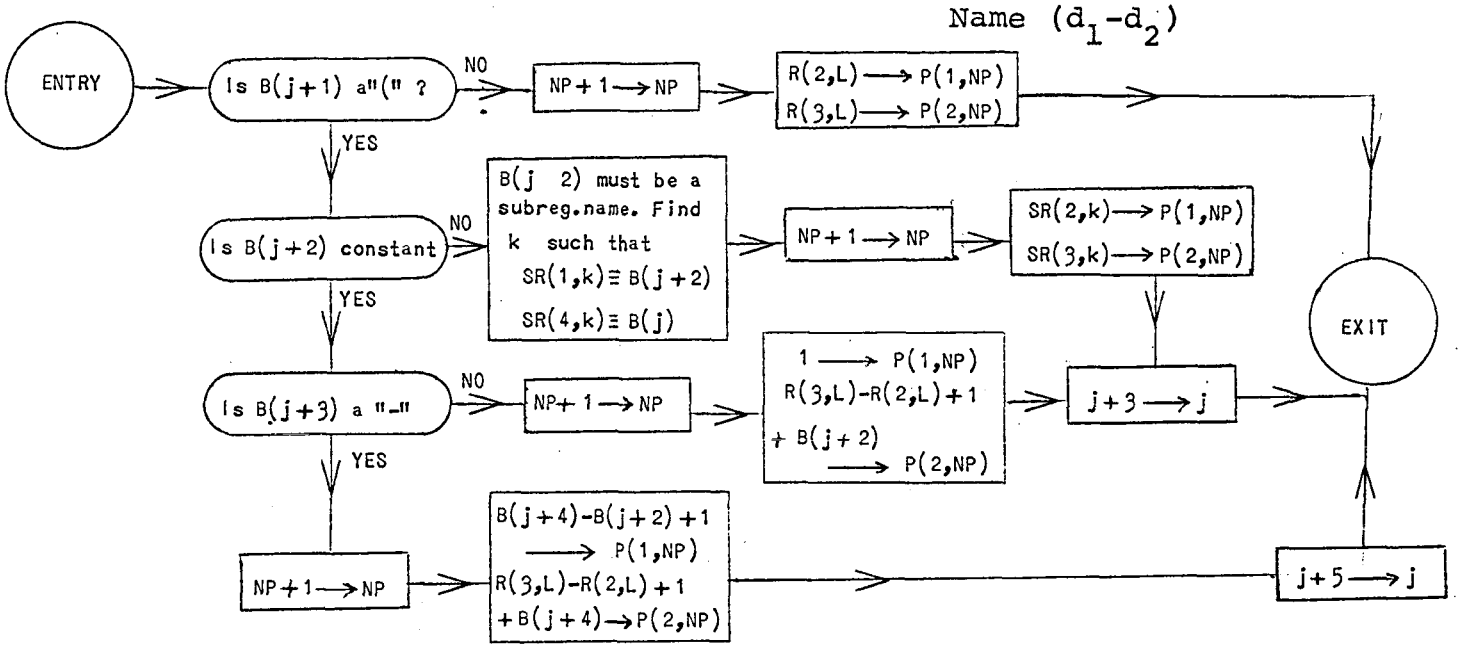
Decode Register, B(j) ≡ R(1,L)

The register may have one of the forms: Name

Name (Subreg. name)

Name ($d_1$)

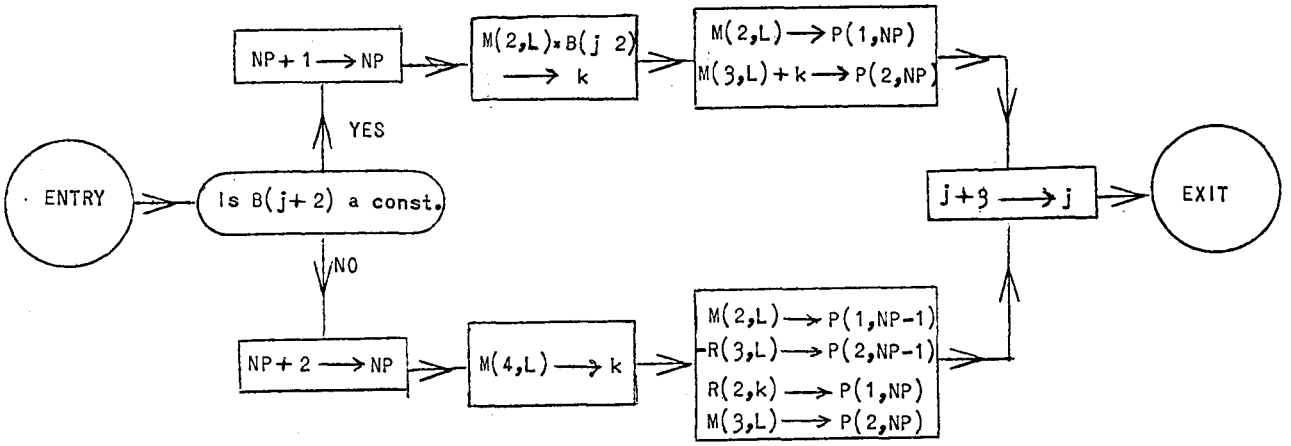
Name ($d_1$-$d_2$)




Decode Memory B(j) ≡ M(1,L)

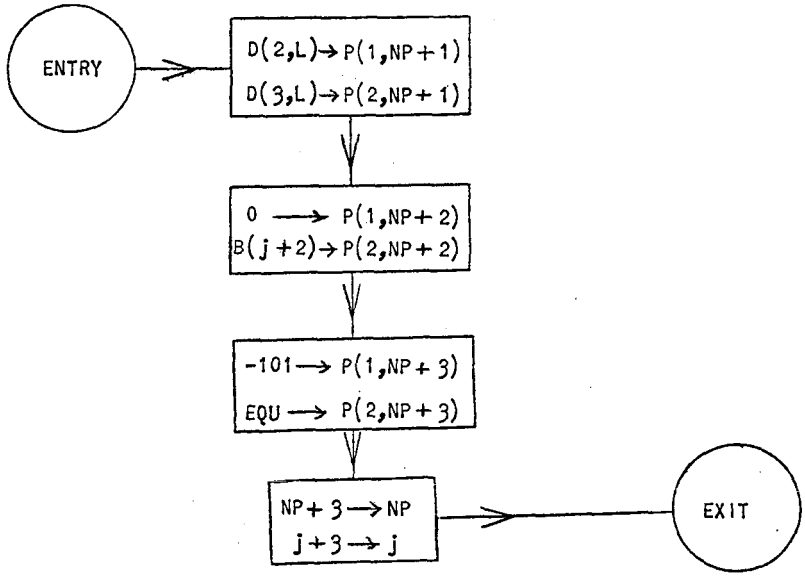
The memory word must have one of the forms: Name (addr. reg.)

Name (constant)



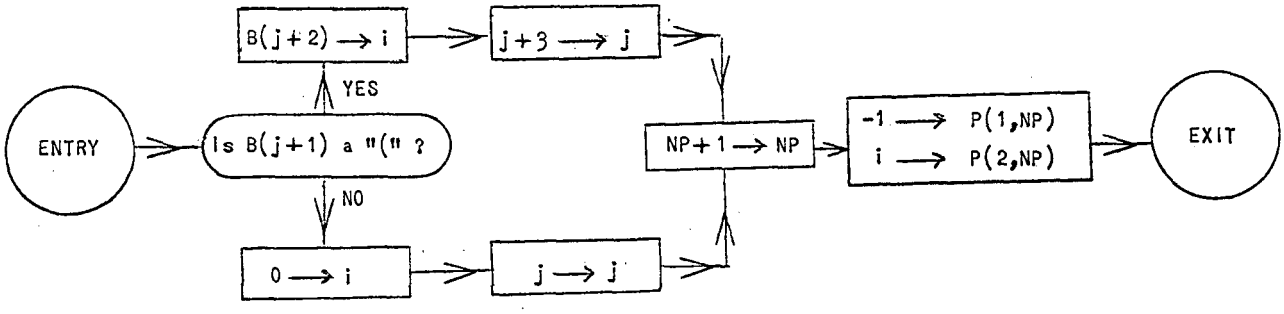
Decode Decoder B(j) ≡ D(1,L)

The Decoder must have the form of: Name (const.)




Decode Clock B(j) ≡ C(1)

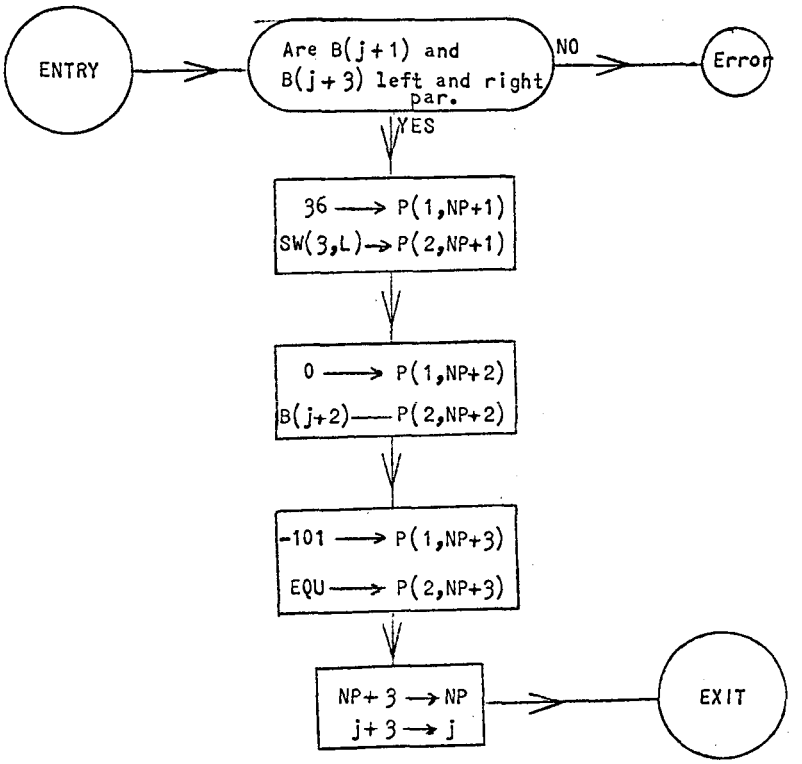
The Clock must have the form of either Name or Name (const.)



Decode Switch B(j), B(j+1), B(j+2), B(j+3) ≡ SW(1,1), (, Pos., )

i.e. It must be an expression NAME (Pos)

which must be translated as YES or NO

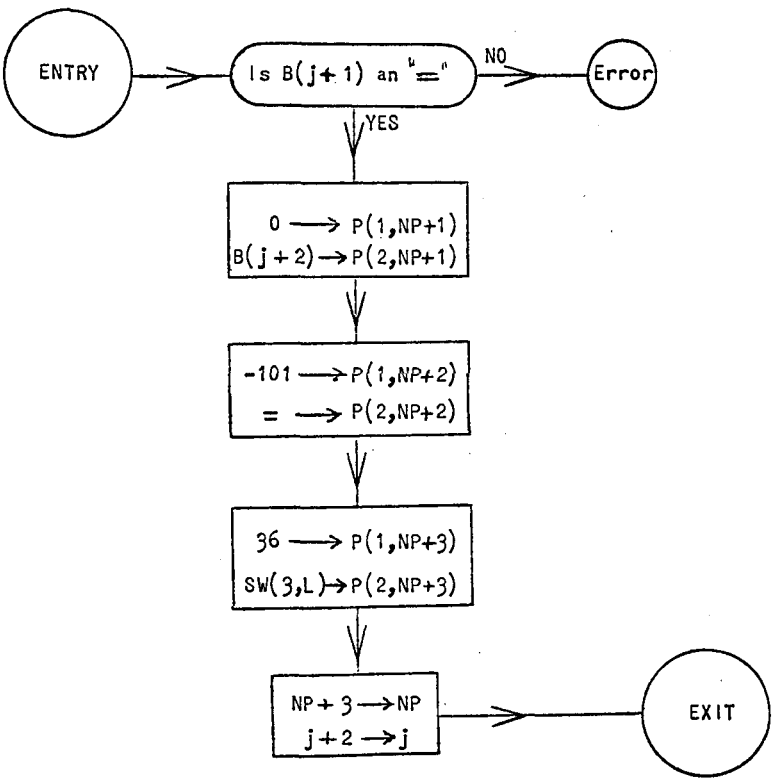


(3e) Translate Switch Setting

This must be a microstatement in the form of

Sw.Name = Sw. position

and B(j) ≡ SW(1,L)



(4) SAVE in Communication Table

This is a part of the Executive Program. The detailed flow charts will be given in another report.

# 4. SIMULATOR

## 4.1 Description

The Simulator executes test programs written for the designed computer and prints out the contents of prescribed registers. Obviously, the calling of the Translator must precede the Simulation because the Simulator needs the results of the translation.

The Simulator consists of the following four routines:

Loader

Output routine

Switch routine

Simulate routine

Loader. For the simulation, it is necessary to read the test program from punched cards and store it into the memory of the designed computer. This is accomplished by the Loader.

Output routine. During the simulation, the contents of certain registers will be printed out. This is handled by the Output routine which must be initialized by the list of the registers whose contents must be printed.

Switch routine. All digital computers have manual switches. The simulation of a manual operation, e.g. turning the POWER switch ON, is accomplished by the Switch routine through reading switch cards.

Simulate routine. The actual execution of the test program is done by the Simulate routine.

The above four routines are not independent. The Simulate routine can call parts of the Output and Switch routines. The actual simulation is done in a loop, called Label Cycle loop. One label cycle consists of 4 steps:

1. The requirement for manual switch setting is tested. If there are switches to set, the corresponding segments of the Polish String are executed.

2. All labels are tested for true or false. No true label causes exit from the loop. The same way, more than one true label corresponding to the same clock time causes an exit.

3. The corresponding microstatements of the true label are executed.

4. The contents of the specified registers are printed by the Output routine.

Control cards

The Simulator has its own control cards for the purpose of initiating the above-mentioned four routines. The control cards have * in column 1, the list starts from col. 10:

```
*LOAD
*OUTPUT    list
*SWITCH    list
*SIM       list
```

The lists are described in the sections of the four routines.

## 4.11 Loads

The Loader is initiated by the control card

col. 1 = *

col. 2-5 = LOAD

The routine assumes that the above control card is followed by data cards containing a test program. These data cards must have the following format:

col. 1 = blank

col. 5-10 = name of a memory of the designed computer, left adjusted, or name of a register.

col. 15-20 = octal location in the memory, right adjusted, blank if col. 5-10 contains a register name.

col. 27-50 = the octal word to be stored in the above defined location, right adjusted.

Comment cards, with C punched in column 1, may be inserted between the test program. The Loader reads the test program card by card, prints it, and stores the octal words in the Storage Array at the defined locations.

4.12 Output routine

The control card, *OUTPUT, initiates the Output routine by defining registers and switches whose contents and positions are to be printed. The list in col. 10-72 may have one of the two forms

LABEL (n) = R1, R2, ..., Rk

CLOCK (n) = R1, R2, ..., Rk

where R1, R2, ..., Rk are the names of registers and switches whose contents and positions are to be printed. In the first case, LABEL (n), the print occurs after every n label cycles; in the second case, CLOCK (n), after every n clock cycles. Naturally, if there was no clock defined in the designed computer, the clock cycle may not be used. n must be a non-zero octal integer.

4.13 Switch routine

The *SWITCH control cards define manual switch settings. The list must have the following form:

col. 10-72: n, SW = PS

4.14 Simulation routine

The control card, *SIM, starts the execution of the test program. Thus the control cards for the Loader, Output and Switch routines must precede it. Since the simulation of the test program may run into an infinite loop, it must be safeguarded by a time limit. This time limit is defined by a maximum number of label cycles allowed and it is given in the list of the control card with one of the 2 forms

col. 10-72 either n

or n (L)

where n and L are nonzero octal numbers.

1. n without the subscript L gives the maximum number of label cycles after which the simulation stops.

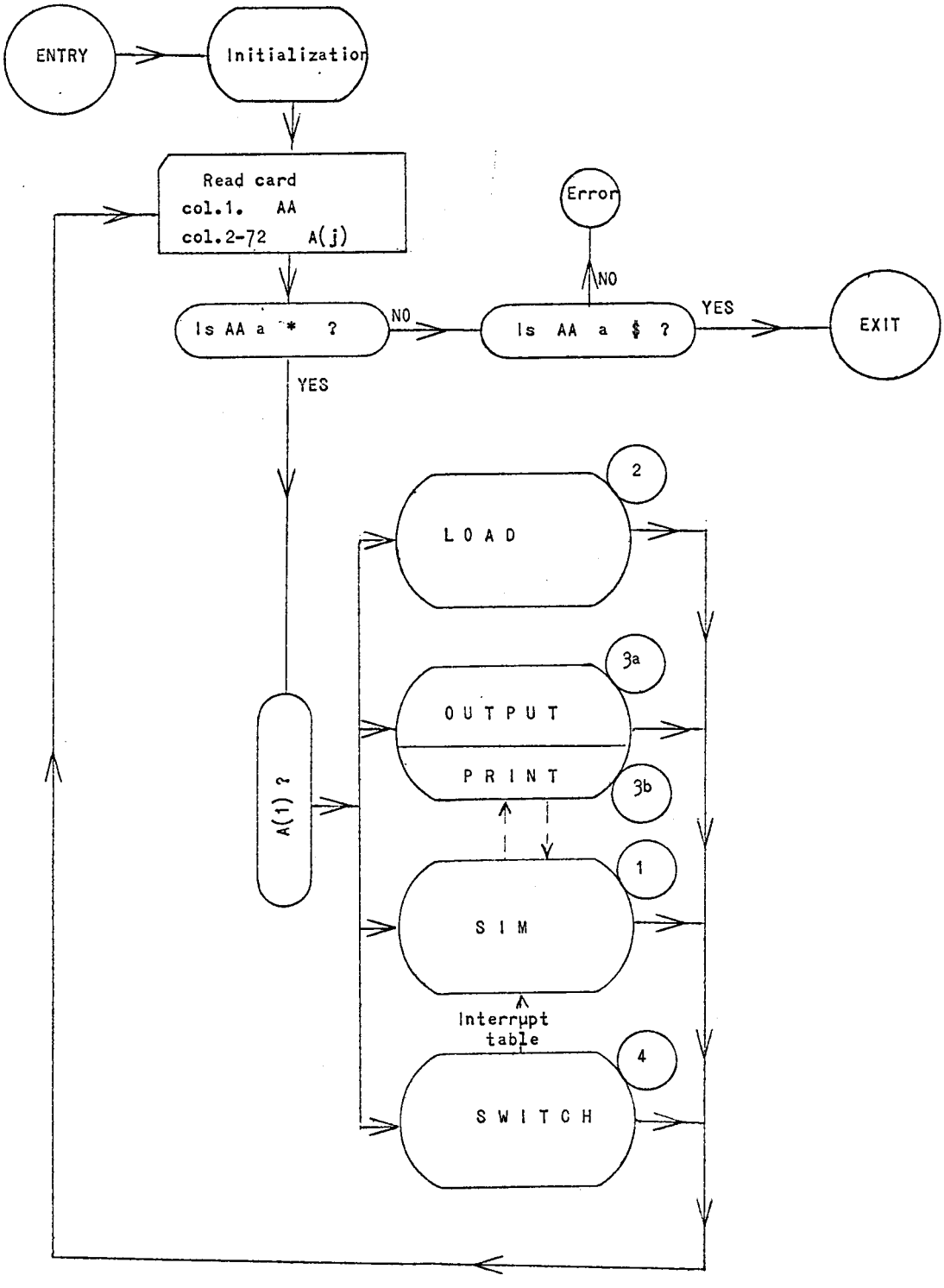
2. If n(L) is given, the simulations stops if the L-th labeled statement is evaluated n-times. The numbering of the labeled statements corresponds to the order as they appeared in the description of the designed computer.

Execution of Segments of the Polish String

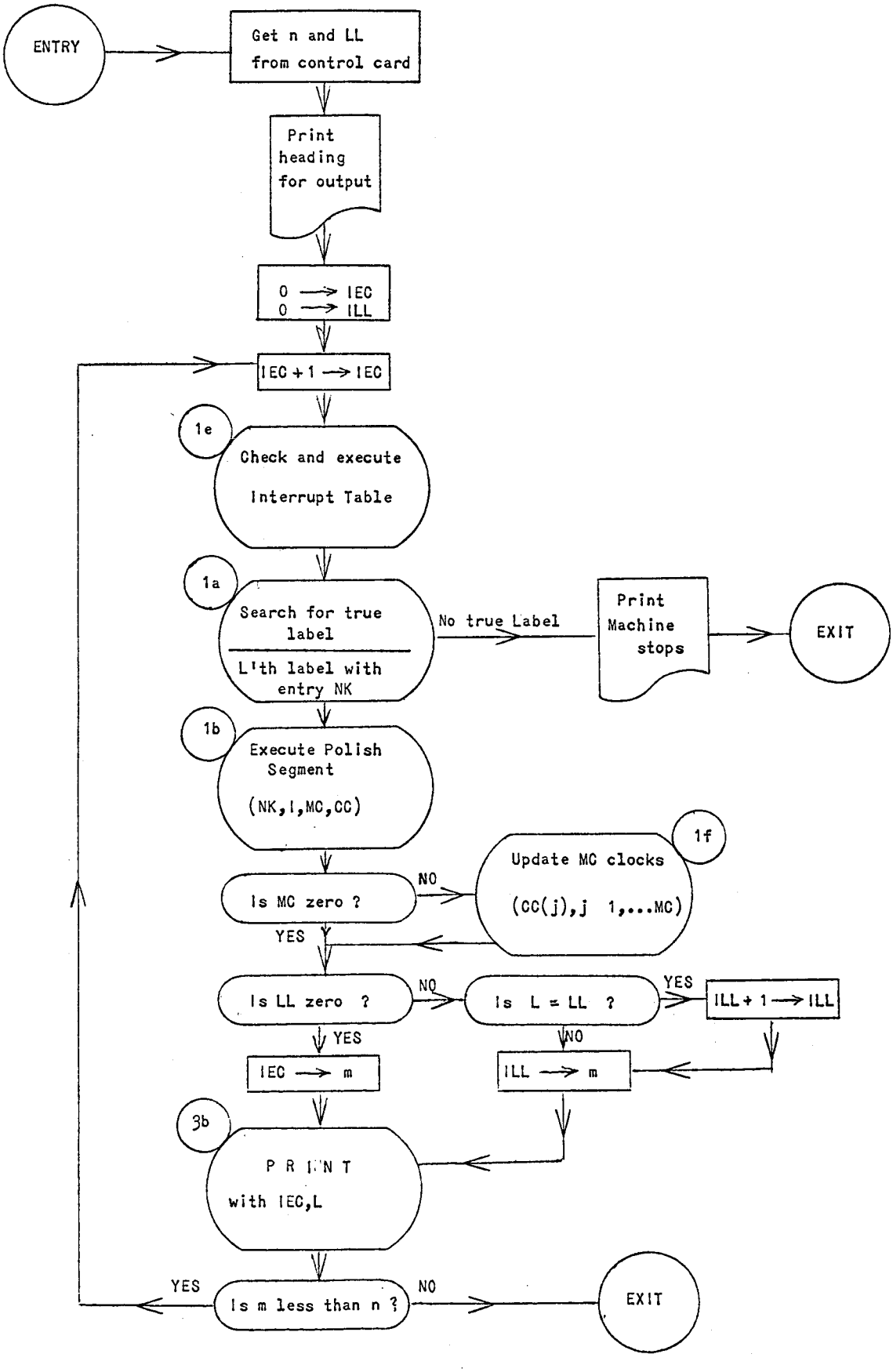
During the simulation, segments of the Polish String must be executed. This is done by the help of a pushed-down pseudo accumulator and by the operator subroutines. The pseudo accumulator (AC) can handle up to 72 bits, thus all registers, memory words, etc., of the designed computer are restricted in maximum size of 72 bits. There are 2 basic subroutines in the Simulator: BRNG (n,L) brings n-bits from location L in the Storage Array into the pseudo accumulator, STRE (n,L) stores the contents of the pseudo accumulator into the Storage Array. The operators (complement, logical OR, AND, functions) must have their corresponding subroutines in the Simulator. The operations are performed between the last two entries of the pseudo accumulator.

## 4.2 SIMULATOR - FLOW CHARTS

In the Initialization, the results of the Translator are received through the Communication Table.

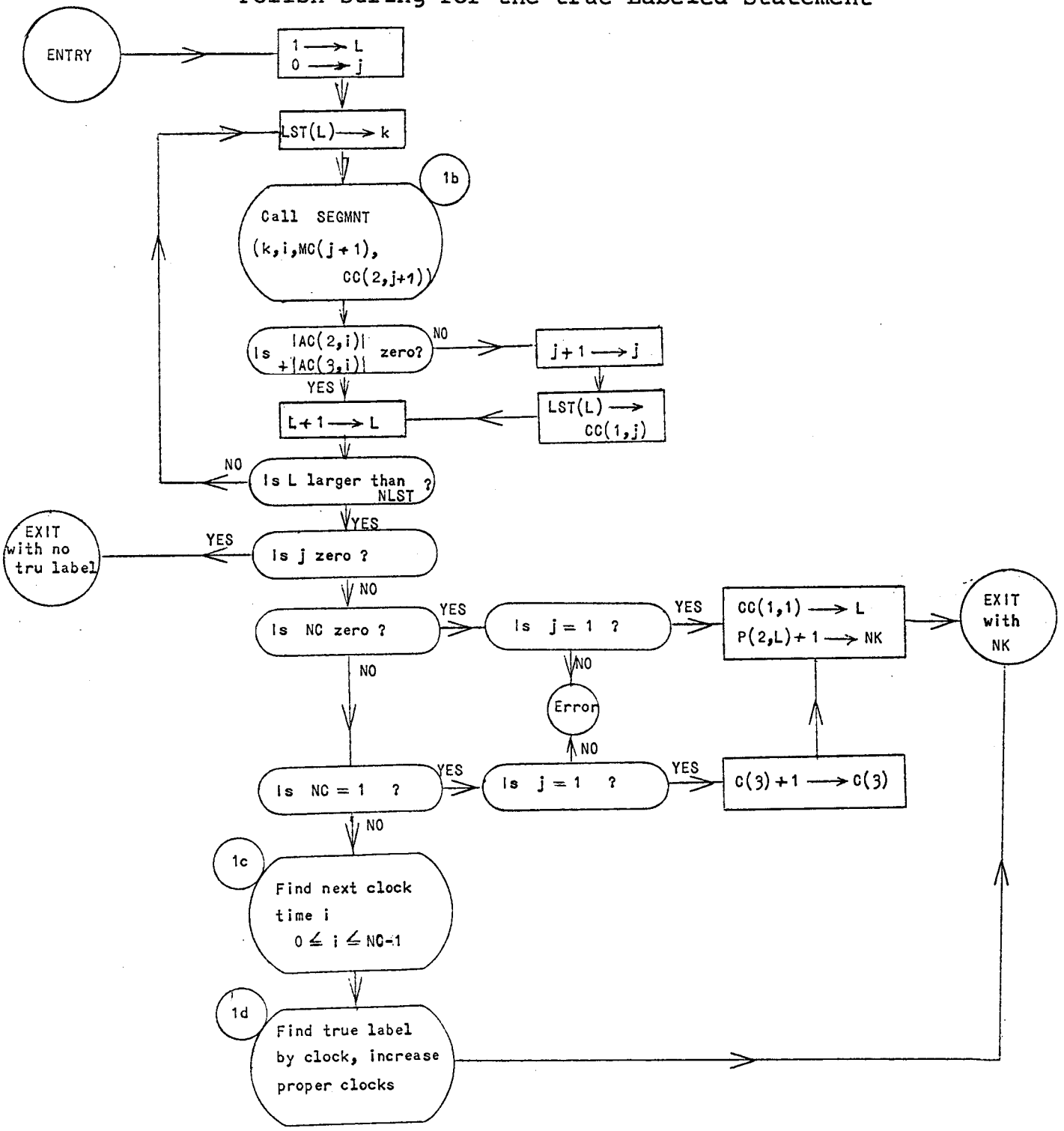


## (1) Simulate routine

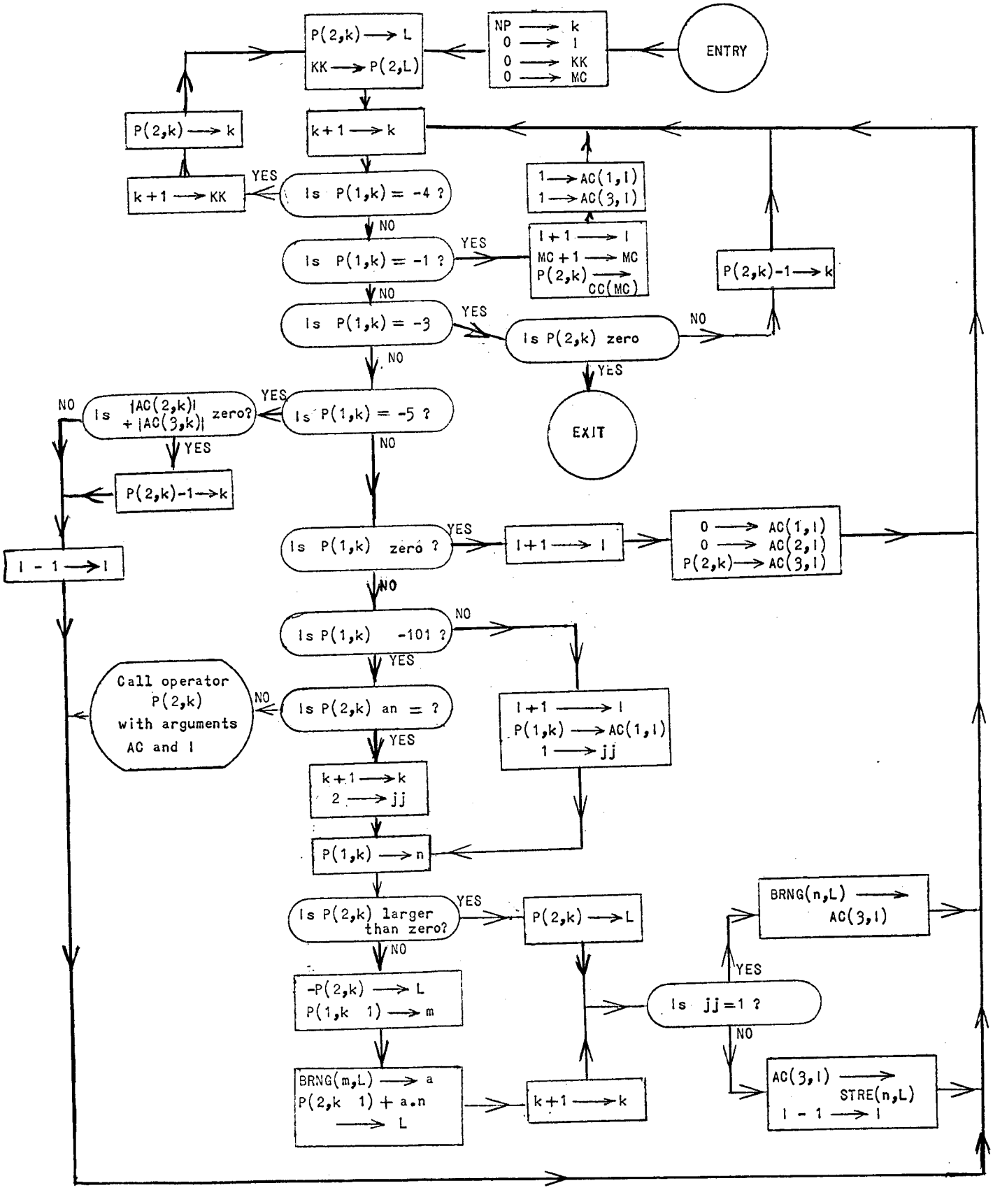


(1a) Search for the unique true Label

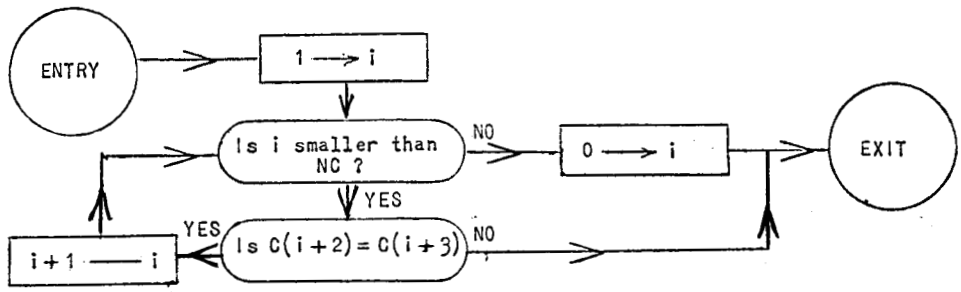
Output: NK = Starting index of the microstatements in the Polish String for the true Labeled Statement



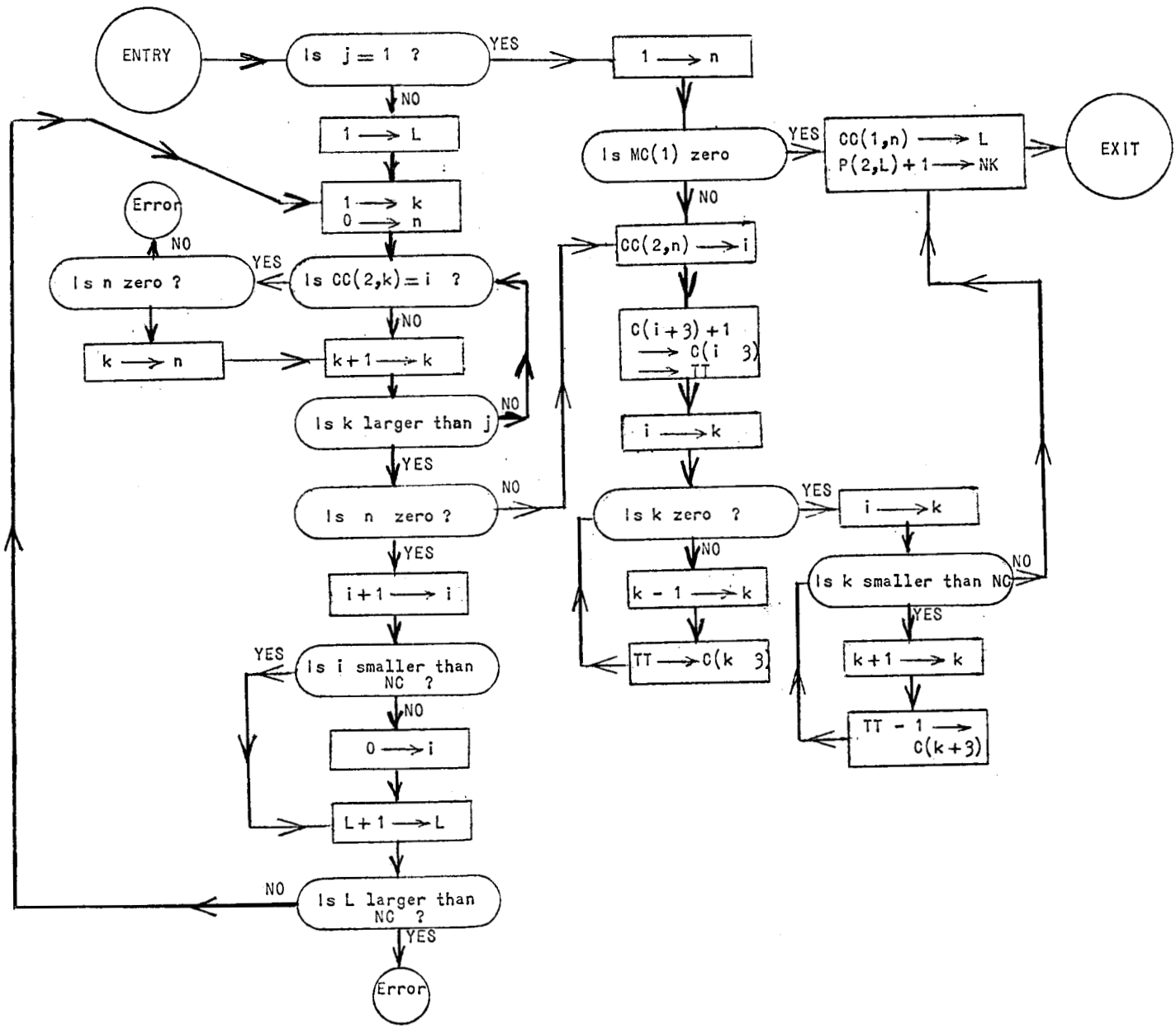
(1b) Execute Polish Sement: SEGMNT (NP, I, MC, CC)



(1c.) Find next clock-time: i

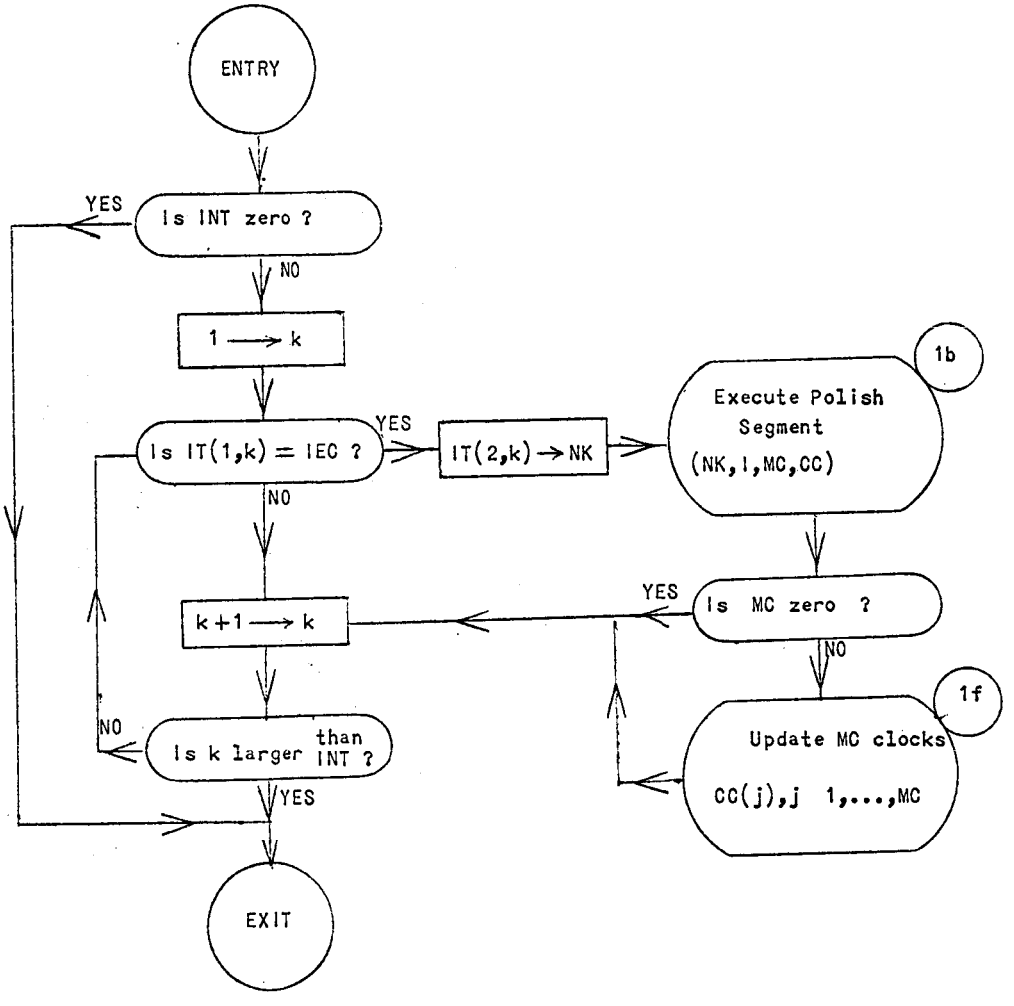



(1d.) Find true Label by clock, increase contents of proper clocks

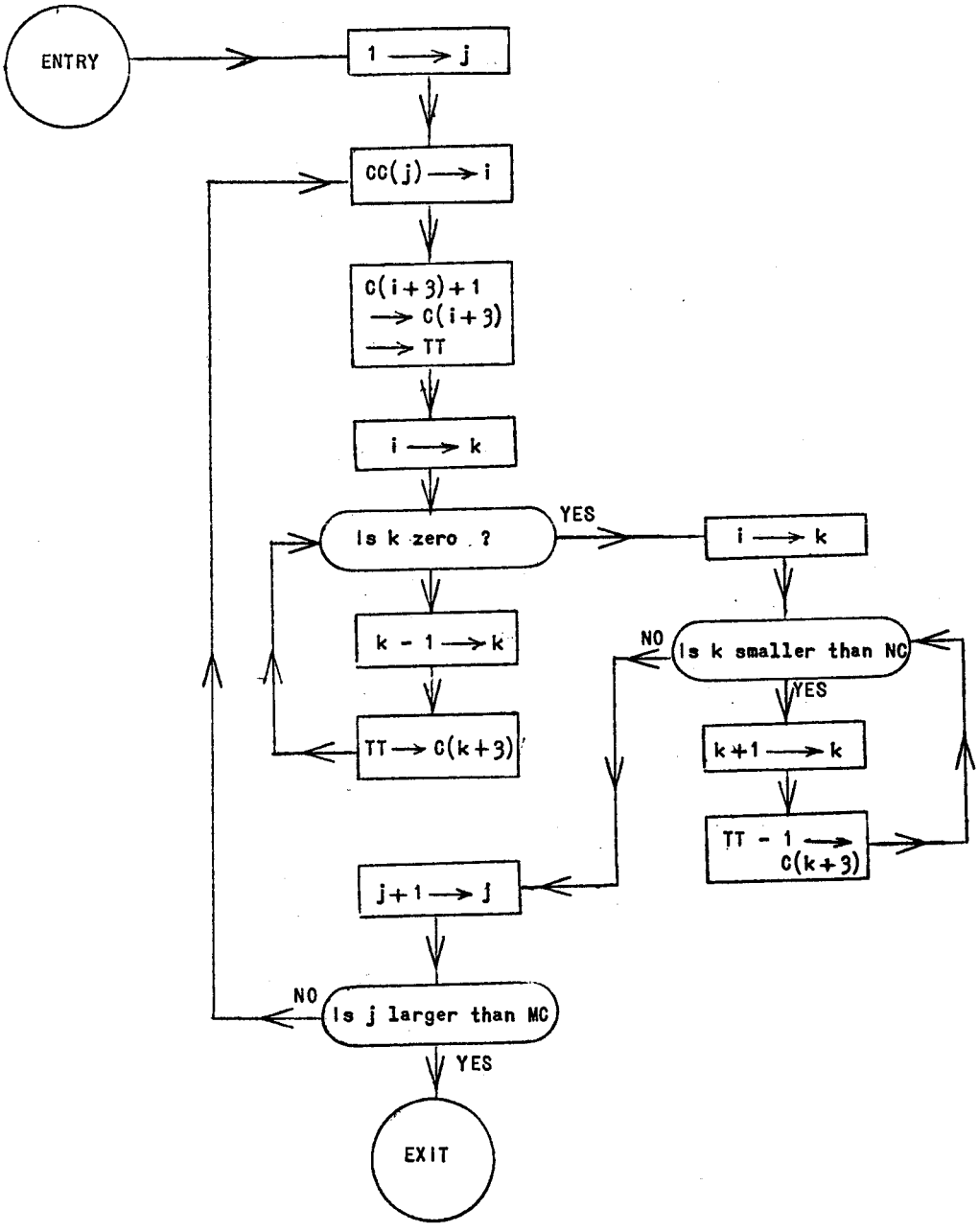


(1e.) Check and Execute Interrupt Table

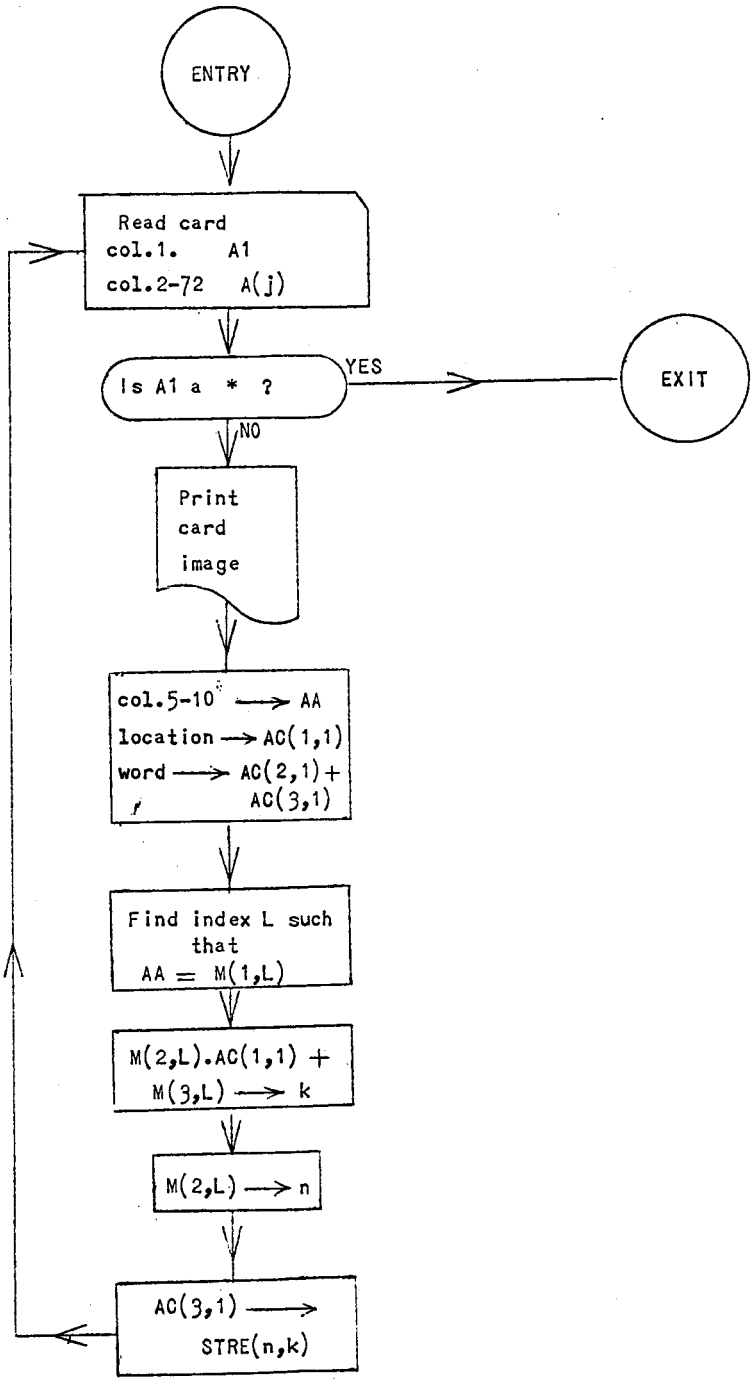
The interrupt table IT(j,k), j = 1, 2, k = 1, ..., INT is established by *SWITCH control cards. IT(1,k) gives the count of the label cycle when the interrupt occurs, IT(2,k) gives the entry-point to the Polish String for the execution of microstatements.



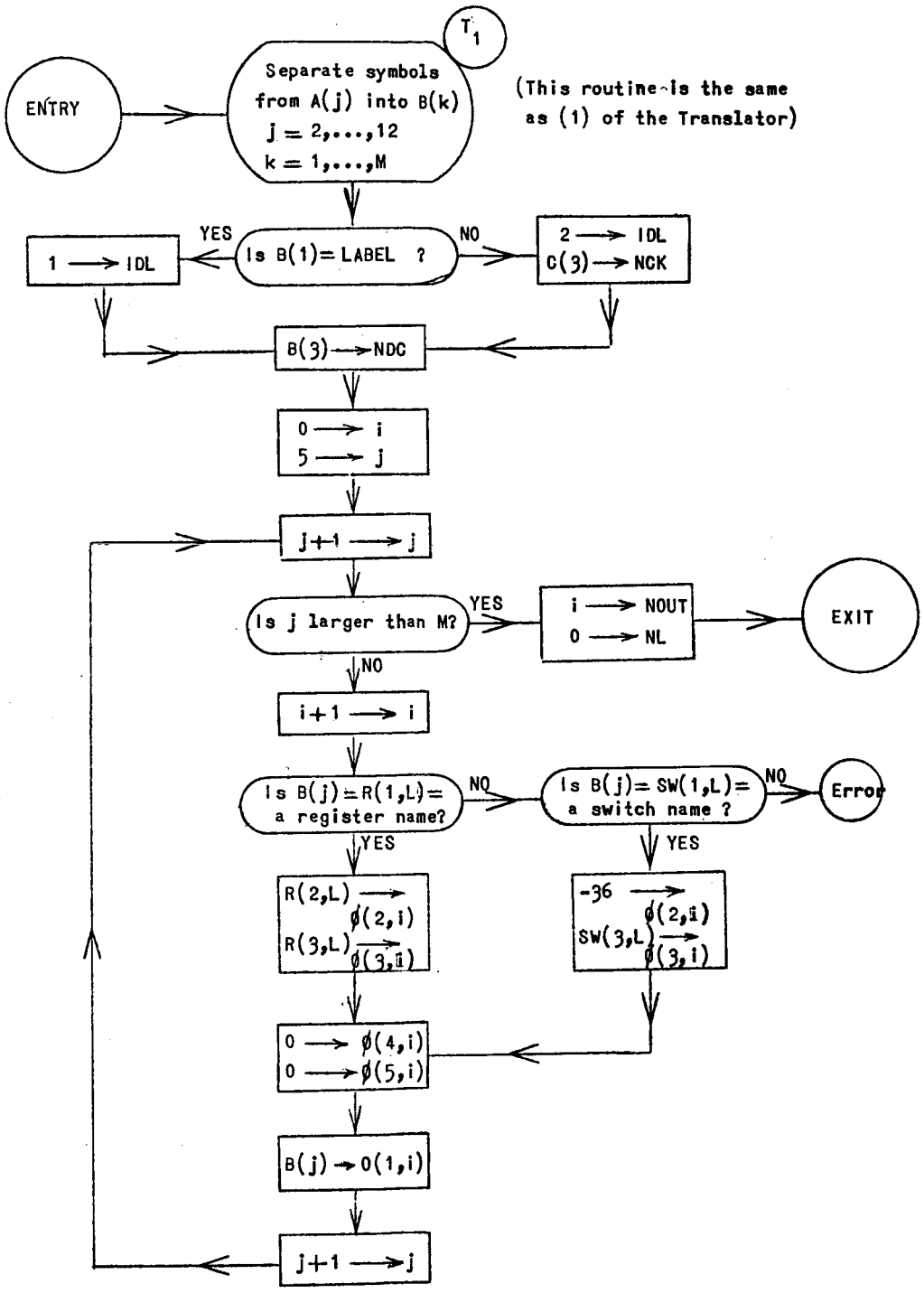
(1f.) Update MC-clocks



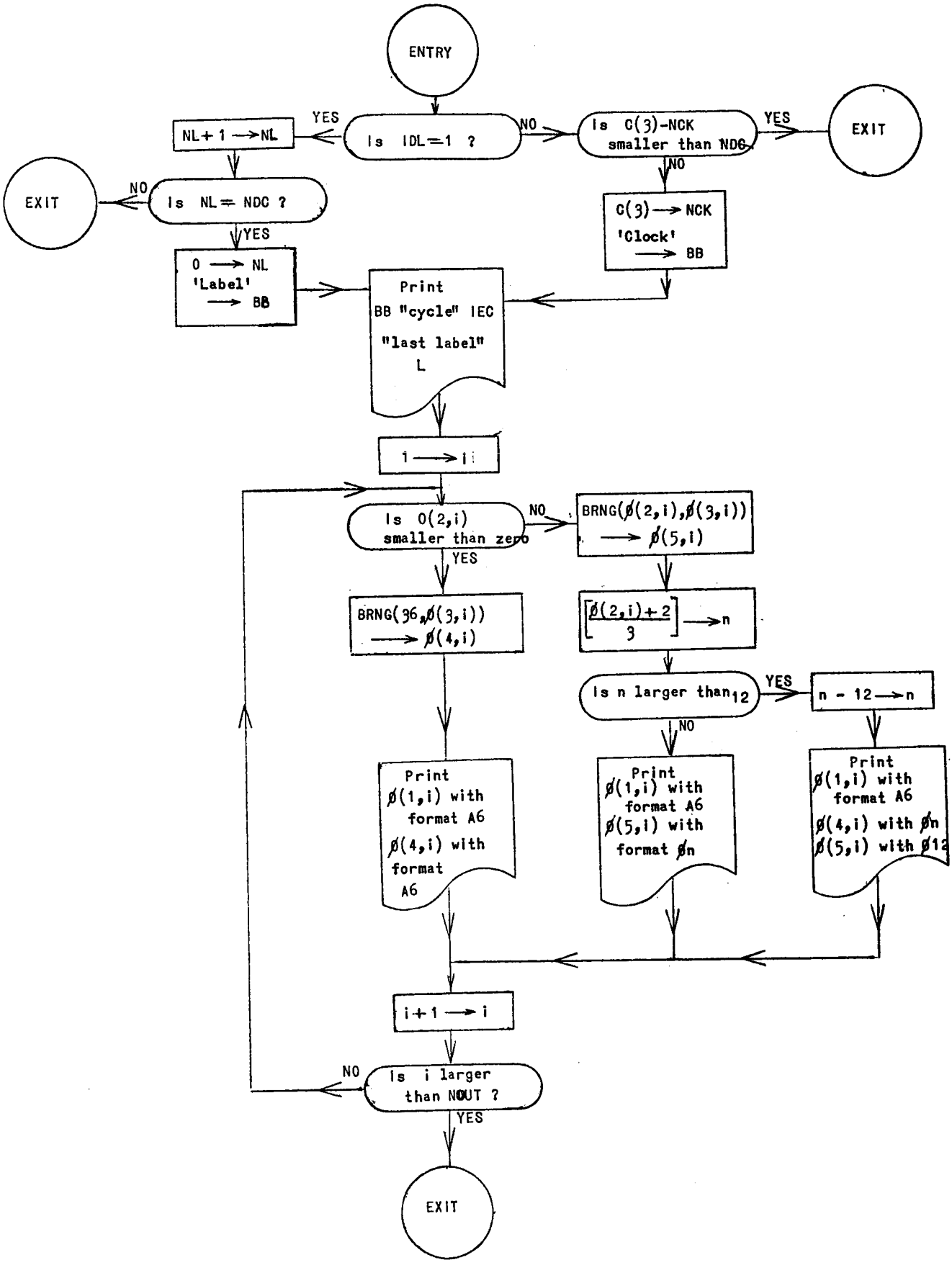
(2) Load routine



(3a.) Initialize Output routine

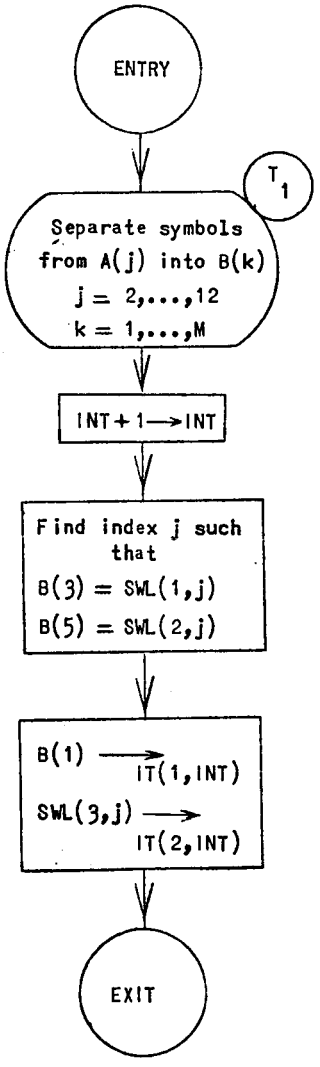


(3b.) Print part of the Output routine



(4.) Switch routine

Interrupt Table: IT(1,j), IT(2,j)




(This routine is the same as (1) of the Translator)

## IV. AVAILABLE BINARY OPERATORS

Form: $A_1$ (operator symbol) $A_2$

| Symbol | Description | Arguments $A_1$, $A_2$ | Number of bits of the result |
|---|---|---|---|
| + | Logical OR | At least one of the arguments must be a variable. If both are variables, they must have the same length in bits: n(A) | n(A) |
| * | Logical AND | At least one of the arguments must be a variable. If both are variables, they must have the same length in bits: n(A) | n(A) |
| - | Exclusive OR | At least one of the arguments must be a variable. If both are variables, they must have the same length in bits: n(A) | n(A) |
| .EQU. | If $A_1$ is equal to $A_2$, the result is one, otherwise zero. | At least one of the arguments must be a variable | 1 |
| .ADD. | The result is the algebraic sum of $A_1$ and $A_2$. The possible overflow bit is discarded. | At least one of the arguments must be a variable. If both are variables, they must have the same length: n(A) | n(A) |
| .SUB. | The result is the algebraic sum of $A_1$, the complement of $A_2$ | At least one of the arguments must be a variable. If both are variables, they must have the same length: n(A) | n(A) |
| .SHR. | $A_1$ is shifted right by $A_2$ bits. | $A_1$ must be a variable with length $n(A_1)$<br>$A_2$ must be a constant | $n(A_1)$ |
| .SHL. | $A_1$ is shifted left by $A_2$ bits. | | |
| .CIRL. | $A_1$ is circled left by $A_2$ bits. | | |

## Information about defining new operators

If one wishes to define new binary operators, there are two steps to be followed:

1. Change the deck OP (ENTRY OPER)
2. Writing the function routine.

1. Assuming the name .XXX. for the new binary operator, the OP routine is changed as follows:

   a) The decimal integer NOP must be increased by one.

   b) BCI 1,XXX

      card must be inserted to the end of the array SYM

   c) TSX XXX,4

      card must be inserted to the end of the array ENT

2. The subroutine XXX($I_1$B) must be provided with the package. The address of B, a(B), provides the addresses of the arguments as follows:

   a(B)-1 and a(B) are the two consecutive addresses where the argument $A_2$ is located in the right adjusted form with preceding zeros.

   a(B)-2 contains the number of bits of $A_2$. It is zero if $A_2$ is a constant.

   a(B)-4 and a(B)-3 are the two consecutive addresses where the the argument $A_1$ is located in right adjusted form with preceding zeros.

   a(B)-5 contains the number of bits of $A_1$. It is zero if $A_1$ is a constant.

   I represents an integer and it must be decreased by one.